Clube Delphi

APIs do Windows de A a Z
Explore as mais diversas APIs

Cadastros e relatórios dinâmicos
Flexibilidade para sistemas

Sistema Financeiro – Parte 3
Implementação da camada DAO

Edição 164 :: R$ 14,90

DEVMEDIA

DELPHI XE 8

O que há de novo

ISSN 1517596-7

MVP

R$ 1.000.000,00
INVESTIDOS EM CONTEÚDO NOS ÚLTIMOS 12 MESES.

APLIQUE ESSE INVESTIMENTO NA SUA CARREIRA...

E MOSTRE AO MERCADO QUANTO VOCÊ VALE!

CONFIRA TODO O MATERIAL QUE VOCÊ TERÁ ACESSO:

- + de **9.000** video-aulas
- + de **290** cursos online
- + de **13.000** artigos
- DEVMEDIA API's consumido + de **500.000** vezes

POR APENAS
R$ 69,90* mensais

*Tempo mínimo de assinatura: 12 meses.

ASSINATURA MVP MOST VALUABLE PROFESSIONAL

PRA QUEM QUER EXIGIR
MAIS DO MERCADO!

DEVMEDIA

Clube Delphi

164ª Edição 2015 ISSN 1517990-7 Impresso no Brasil

Editor Geral

Paulo Quicoli (pauloquicoli@gmail.com)

Equipe Editorial

Guinther Pauli (guintherpauli@gmail.com)

Fabrício Hissao Kawata (fabricio.kawata@bol.com.br)

Giuliano Scombatti Pinto (giuliano@sygnux.com.br)

Daniel Sobrinho Laporte (daniel.laporte@gmail.com)

Jornalista Responsável

Kaline Dolabella - JP24185

Consultor Técnico

Daniella Costa (daniella.devmedia@gmail.com)

Capa e Diagramação

Romulo Araujo

Atendimento ao Leitor

A DevMedia conta com um departamento exclusivo para o atendimento ao leitor. Se você tiver algum problema no recebimento do seu exemplar ou precisar de algum esclarecimento sobre assinaturas, exemplares anteriores, endereço de bancas de jornal, entre outros, entre em contato com:

www.devmedia.com.br/central
(21) 3382-5038

Publicidade

Para informações sobre veiculação de anúncio na revista ou no site e para fechar parcerias ou ações específicas de marketing com a DevMedia, entre em contato com:

publicidade@devmedia.com.br

Distribuição

FC Comercial e Distribuidora S.A
Rua Teodoro da Silva, 907
Grajaú - RJ - 206563-900

Fale com o Editor!

É muito importante para a equipe saber o que você está achando da revista: que tipo de artigo você gostaria de ler, que artigo você mais gostou e qual artigo você menos gostou. Fique a vontade para entrar em contato com os editores e dar a sua sugestão!

Se você estiver interessado em publicar um artigo na revista ou no site ClubeDelphi, entre em contato com os editores, informando o título e mini-resumo do tema que você gostaria de publicar:

Paulo Quicoli - Editor da Revista
pauloquicoli@gmail.com

Paulo Quicoli - **Editor Geral**

pauloquicoli@gmail.com
quicoli.wordpress.com
twitter.com/pauloquicoli

MVP

Assine agora e tenha acesso a todo o conteúdo da DevMedia: **www.devmedia.com.br/mvp**

DEVMEDIA

Sumário

Conteúdo sobre Novidade

04 – Conheça as novidades no Delphi XE 8
[*Fabricio Hissao Kawata*]

Artigo no estilo Curso

13 – Desenvolvendo um Sistema Financeiro em Delphi – Parte 3
[*Filipe Dalepiane*]

Artigo no estilo Curso

22 – Explorando APIs do Windows em Delphi – Parte 1
[*Vanderson Cavalcante de Freitas*]

Artigo no estilo Solução Completa

34 – Cadastros e relatórios dinâmicos em Delphi
[*Vanderson Cavalcante de Freitas*]

Dê seu feedback sobre esta edição!

A ClubeDelphi tem que ser feita ao seu gosto. Para isso, precisamos saber o que você, leitor, acha da revista!

Dê seu voto sobre esta edição, artigo por artigo, através do link:

www.devmedia.com.br/clubedelphi/feedback

Conheça as novidades no Delphi XE 8

Fique por dentro das principais novidades desta versão

Dois anos de aperfeiçoamentos trouxeram o Dephi XE8, que logo de início, apresenta como primeiro ponto de grande impacto e mudança a própria aparência do IDE, como podemos conferir na **Figura 1**. Com um visual rantes, sem perder seu aspecto singular. Com um visual renovado a ferramenta ganha um look and feel inédito, com fontes de texto nativamente maiores, cores mais vibrantes, sem perder seu aspecto singular. Para muitos críticos esta pode ser interpretada como sendo uma pontualidade sem expressão, mas o fato é que essa simples mudança vem a afirmar a constante evolução do Delphi como produto, o que garante a manutenção de seus adeptos, bem como a conquista de novos

Fique por Dentro

O novo Delphi traz para a sua comunidade uma série de novidades, refletidas em incorporações de novos elementos, integrações nativas com novas ferramentas, como o AppAnalytics, suporte a novas tecnologias, como IoT, bem como melhorias pontuais nos mais diversos aspectos do produto.

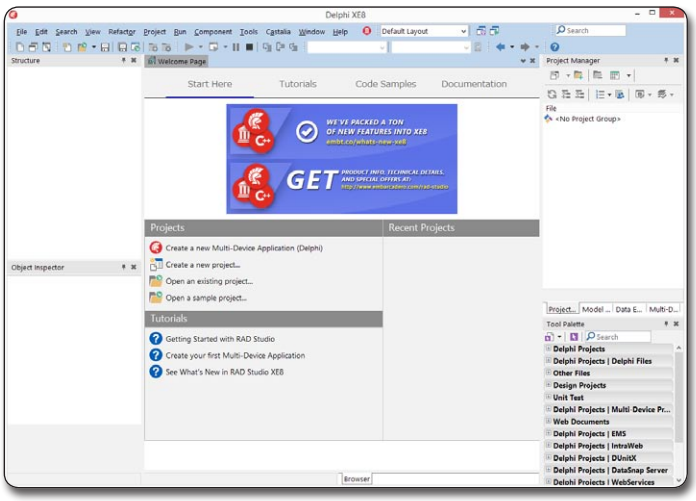


Figura 1. IDE do Delphi XE8

Novidades na VCL

Como não poderia deixar de ser, a VCL (Visual Component Library), tida como a parte clássica do Delphi, conta com algumas novidades exclusivas nesta nova versão do IDE. Tendo em vista sua estabilidade, aliada a enorme expansão do FireMonkey no cenário mais recente, poucas são suas novas pontualidades nesta versão, mas podemos destacar a facilidade com conexões de Internet ou Bluetooth com componentes de AppTethering.

Novos VCL Styles

Surgido como uma das grandes novidades do Delphi XE2, o VCL Styles pode ser brevemente definido como sendo um recurso para incrementar a aparência de uma aplicação VCL. Para isso, são utilizados estilos, cada qual definindo um conjunto elaborado de detalhes gráficos personalizados que incidirão sobre o look and feel da aplicação. Logo, sua atuação ocorre de forma semelhante ao tradicional recurso de temas do Windows.

Visando manter o valor das aplicações construídas e constante melhora para novos desenvolvimentos VCL, o Delphi XE8 apresenta agora mais três novos estilos para VCL Styles, intitulados Glow, Sky e Tablet Light. O ganho aqui fica por conta da notória evolução de um recurso interno que, anteriormente ao XE2, só era conseguido por meio do uso de componentes de terceiros, o que acabava por gerar uma dependência natural para com estes, além de eventualmente dispender custos para aquisição e uso.

AppAnalytics

O AppAnalytics pode ser considerado um novo recurso disponível para projetos VCL e FireMonkey, que prevê a coleta de dados (de forma anônima) resultantes da interação do usuário com a aplicação, possibilitando então uma auditoria de processos por meio da interpretação e análise dessas informações.

Copyright - Proibido copiar ou distribuir. Todos os direitos reservados para DevMedia

No Delphi, o AppAnalytics se reflete em um componente homônimo **(Figura 2)**, oriundo da classe TAppAnalytics.

Na prática, os dados capturados pelo componente são então enviados a um aplicativo Web AppAnalytics para imediato processamento e eventual análise. Por indiretamente lidar com informações de uso e envolvimento do usuário, a própria Embarcadero faz questão de ressaltar em caixa alta a forma anônima que o componente atua, se restringindo a transmitir as informações somente ao contexto da aplicação em questão.

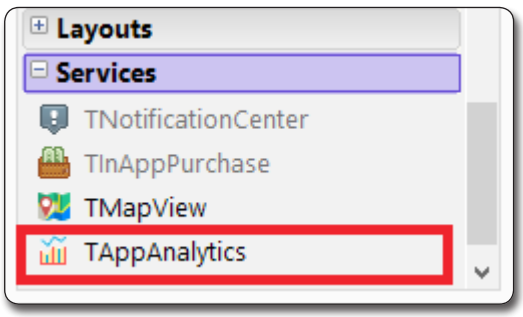


Figura 2. Componente TAppAnalytics

Novidades para FireMonkey

O Delphi XE8 traz consigo também uma série de melhorias para o FireMonkey. Grande parte delas trata de pequenos aspectos sutis, mas que não deixam de impactar de forma direta em diversos pontos relacionados a um projeto Mobile Application.

TWebBrowser

O TWebBrower é um componente visual do FireMonkey que, uma vez adicionada à aplicação, toma a forma de um Web Browser interno permitindo que este carregue e mostre conteúdos Web em sua área. Inicialmente introduzido na ferramenta como sendo de uso exclusivo às construções Mobile (Android e iOS), agora no XE8 seu uso é expandido às construções Desktop (Windows e Mac OS X).

Apresentação nativa de diversos controles no iOS

Nesta nova versão do IDE, especificamente para projetos de Target Platform iOS, os tradicionais controles TCalendar, TEdit, TListView, TMemo, TMultiView e TSwitch ganham agora uma forma de apresentação nativa (native), em complemento a sua já existente apresentação estilizada (styled). O impacto direto pela escolha de uma ou outra forma de apresentação se dá então sobre o aspecto visual do controle, fazendo com que o mesmo assuma uma representação nativa da própria plataforma de destino, quando marcado como native, ou siga uma estilização do próprio FireMonkey, no caso de styled. Na prática, essa escolha se dá pela configuração da propriedade ControlType dos controles, conforme mostrado na **Figura 3**.

Sem deixar de citar, apesar da novidade, styled ainda se mantém como o valor padrão para a propriedade.

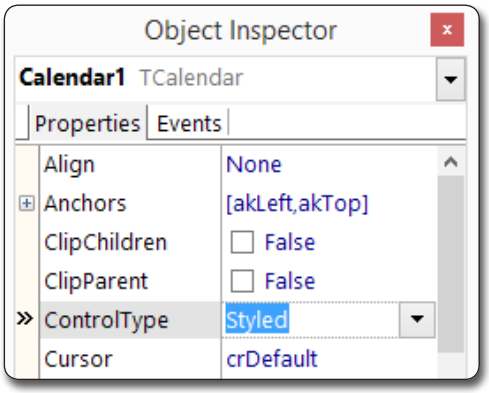


Figura 3. Propriedade ControlType

TMapView

Agora no XE8 os aplicativos para Android e iOS com FireMonkey oferecem suporte ao componente TMapView, cuja função essencial é prover interatividade com os mapas dessas aplicações por meio de ações como: controle por gestos (gestures), adição de marcadores (markers), controle de coordenadas e orientação. Adicionalmente, ainda por meio do uso de um objeto MapView é possível determinar o estereótipo do mapa por entre quatro segmentações: normal, satélite, híbrido e terreno.

Aqui um ponto de observação: em aplicativos para Android, o uso do TMapView requer a obtenção de um certificado adicional da própria Google, denominado Google Maps API Key para a efetiva utilização das funcionalidades envolvidas. Caso isso não seja feito, tal pontualidade acaba por se tornar um fator determinante, uma vez que em tempo de execução uma exceção é gerada, inviabilizando o uso ideal da aplicação. Esse certificado é gerado usando o login no Google.

Photo Library

De olho na melhor usabilidade e experiência do usuário em suas aplicações móveis, o FireMonkey conta agora com um recurso que possibilita salvar fotos capturadas pela câmera do dispositivo diretamente na biblioteca de fotos do sistema do próprio aparelho.

Na prática, a utilização desta nova opção se dá pelo envolvimento e manuseio de alguns elementos distintos. O primeiro a ser citado é o IFMXPhotoLibrary, que caracteriza uma interface para salvar as imagens no álbum de fotos do sistema. Para isso, sua estrutura conta com um método exclusivo denominado intuitivamente de AddImageToSavedPhotosAlbums, que acaba por dispensar comentários. Relacionado a isso surge então outros dois elementos, TParamsPhotoQuery e IFMXCameraService, ambos relacionados à obtenção de fotos a partir da câmera. Por fim, a propriedade NeedSaveToAlbum é que determina se a aplicação poderá ou não salvar as fotos obtidas pela câmera na biblioteca de fotos.

Standard Actions para TMediaPlayer

Na FMX, TMediaPlayer é o componente responsável pela manipulação de arquivos multimídia, garantindo a devida interação

Copyright - Proibido copiar ou distribuir. Todos os direitos reservados para DevMedia

da aplicação com formatos de áudio e vídeo. Logo, neste novo release, o componente Action List (TFMXActionList) acaba por ganhar novos acréscimos de Standard Actions, agora relacionadas a TMediaPlayer, aumentando ainda mais a lista de ações pré-definidas disponibilizadas ao desenvolvedor. Num total de cinco ações, três descendem de TCustomMediaPlayerAction e cada uma acaba por determinar um comportamento singular do player:

- TMediaPlayerStart: inicia (start) ou continua (resume) a reprodução da mídia;
- TMediaPlayerStop: interrompe a reprodução da mídia;
- TMediaPlayerPlayPause: inicia ou determina uma pausa na reprodução da mídia.

Na mesma medida, as outras duas são originárias de TMediaPlayerValue e lidam com valores de propriedades do player:

- TMediaPlayerCurrentTime: provê controle sobre o tempo atual (propriedade CurrentTime do player) da mídia;
- TMediaPlayerVolume: provê controle sobre o volume (propriedade Volume do player) da mídia.

Form Positioner

O Form Positioner é um recurso presente no Delphi que, conforme seu próprio nome sugere, está voltado à definição do posicionamento dos formulários da aplicação. Já presente há bastante tempo no IDE para projetos do tipo VCL Forms, sua representação visual incide em um ícone localizado na parte inferior direita do Form Designer do IDE (**Figura 4**), denominado Form Positioner Preview. Por meio de seu acionamento é possível então determinar a posição que será adotada pelo Form quando a aplicação estiver em execução (run-time).

Dito isto, agora no XE8, a novidade neste quesito fica por conta da disponibilidade deste recurso também para projetos FireMonkey, isto porque, conforme já mencionado, o mesmo era restrito a construções VCL.

ImageList

Já bastante tradicional na VCL, o ImageList é um componente centralizador de pequenas imagens, formando assim uma coleção que servirá de aporte aos diversos elementos de interface visual (GUI) de uma aplicação, tais como menu, botões, etc. Seguindo por esta mesma linha, nesta versão do IDE, enfim o FireMonkey ganha o seu TImageList, cuja proposta de uso se assemelha ao ImageList habitual da VCL.

Embora parecido, o componente da FMX acaba por apresentar recursos adicionais se comparado ao seu coirmão. O principal deles diz respeito ao seu trabalho automático com bitmaps de várias resoluções (multi-resolution bitmaps), o que garante a apropriada visualização da imagem nas mais distintas resoluções em que a aplicação poderá ser executada. Tal comportamento acaba por se tornar essencial, tendo em vista a característica multi-plataforma do FireMonkey, que implica sua atuação em muito mais cenários que a VCL.

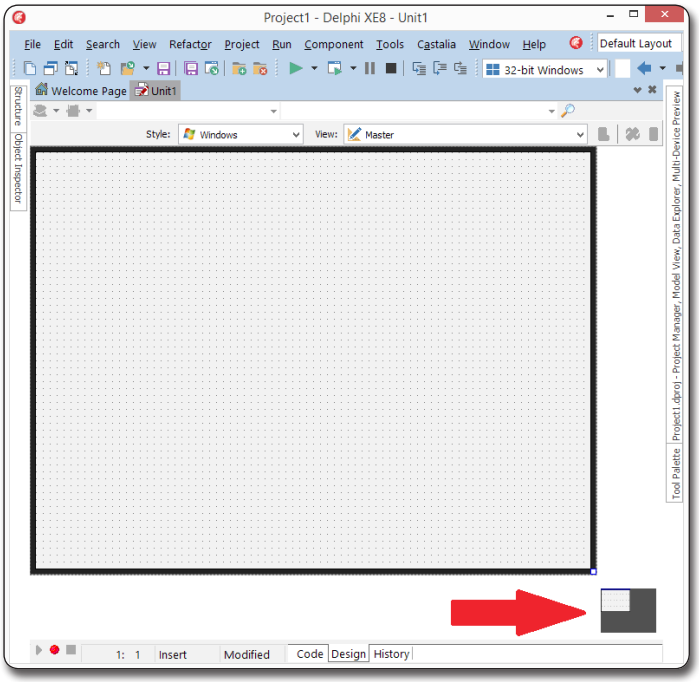


Figura 4. Propriedade ControlType

CaptureSettings

Como já é sabido, TCameraComponent é o componente de câmera da FMX. Em sua utilização, a propriedade Quality é que define a qualidade da captura. Até a versão XE7 essa definição se dava pela escolha entre quatro opções, cada qual referente a um nível de resolução distinta: PhotoQuality, HighQuality, MediumQuality e LowQuality.

Olhando pelo lado técnico, a declaração da propriedade citada ocorre como a seguir:

```
property Quality: TVideoCaptureQuality read GetQuality write SetQuality;
```

Logo, seu tipo TVideoCaptureQuality nada mais é que um enumerado, cuja a estrutura até a versão anterior da ferramenta pode ser vista a seguir:

```
TVideoCaptureQuality = (PhotoQuality, HighQuality, MediumQuality, LowQuality);
```

Agora no Delphi XE8 essa mesma estrutura conta com uma pequena alteração, com a adição de um novo elemento denominado CaptureSettings:

```
TVideoCaptureQuality = (PhotoQuality, HighQuality, MediumQuality, LowQuality,
CaptureSettings);
```

Como pode se presumir, CaptureSettings abre então mais uma opção ao desenvolvedor com relação à definição da qualidade de captura do componente de câmera para projetos FireMonkey. Por definição, ela provê uma forma customizada de resolução.

DEVMEDIA Copyright - Proibido copiar ou distribuir. Todos os direitos reservados para DevMedia

Novidades na RTL

Como não poderia deixar de ser, o core do Delphi também ganha melhorias neste novo release da ferramenta. Sendo assim, agora no XE8 a RTL (Run-Time Library) ganha o acréscimo de novas units, o que resulta em novos recursos e componentes abrangentes aos dois contextos de desenvolvimento do IDE: VCL e FMX.

TNetHTTPClient e TNetHTTPRequest

Conforme mostra a **Figura 5**, TNetHTTPClient e TNetHTTPRequest são dois novos componentes que surgem na RunTime Library do Delphi. De forma básica a usabilidade de ambos está totalmente inter-relacionada, uma vez que lidam diretamente com questões cliente-servidor do contexto HTTP. Mais especificamente, por meio de seus recursos (métodos, propriedades e eventos) torna-se facilitada as eventuais tratativas de pedidos de request cliente, tal qual a ação de resposta (response) oriunda do servidor HTTP.

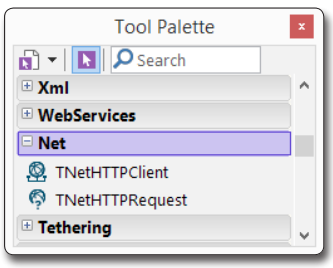


Figura 5. Componentes TNetHTTPClient e TNetHTTPRequest

Suporte a Beacon

No cenário tecnológico mais recente, a palavra beacon é um termo empregado para designar um dispositivo Bluetooth Low Energy. Esta tecnologia nada mais é do que um tipo de Bluetooth específico, que pode incluir informações em seus dados de publicidade (advertising) as quais possibilitarão sua identificação e cálculo de sua distância por qualquer outro dispositivo, dentro de uma margem de área, sem que ele esteja efetivamente conectado ou emparelhado ao dispositivo Beacon (Beacon device).

Visando atender a mais este cenário de possibilidades, o Delphi XE8 inclui na RTL uma API Beacon exclusiva, restrita a projetos para Mac OS X, iOS e Android. Por questões de acessibilidade, a plataforma Windows não é suportada. Além disso, até o presente momento, dois são os tipos Beacon suportados:

- iBeacon: formato proprietário definido pela Apple, tido como o modo "padrão";
- AltBeacon: formato aberto. Também conhecido como Alternative Beacon e, portanto, tido como modo "alternativo" pela ferramenta.

O ponto de vista prático deste suporte a Beacon no Delphi acaba por envolver diversas nuances, o que dá margens à elaboração de um artigo exclusivo deste contexto. Todavia, apenas para citar, o início da efetiva utilização do recurso ocorre por meio da obtenção de uma instância de TBeaconManager, tal como mostrado a seguir:

```
BeaconManager := TBeaconManager.GetBeaconManager(TBeaconScanMode.Standard);
```

TBluetooth

Ainda relacionado a Bluetooth, o XE8 conta com a adição de um novo componente em sua Tool Palette (**Figura 6**) denominado TBluetooth. Sua função principal é agir como um wrapper das classes do Bluetooth "clássico" do framework do Delphi, tais como TBluetoothManager e TBluetoothDevice, encapsulando assim os principais recursos das mesmas. Com essa simples mudança a Embarcadero novamente foca no aspecto RAD que consagrou a ferramenta, tornando possível uma abordagem totalmente baseada em componentes por parte do desenvolvedor.

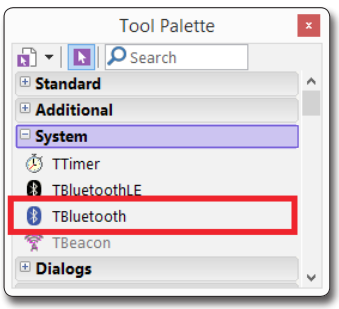


Figura 6. Componente TBluetooth

Hash API

A RTL traz ainda uma nova API Hash, centrada na também nova unit System.Hash, que inclui toda uma estrutura de classes e métodos que habilitam o uso direto de três funções Hash na construção de aplicações Delphi: MD5 (THashMD5), SHA-1 (THashSHA1) e Bob Jenkins (THashBobJenkins). A fim de facilitar sua utilização, todas acabam por apresentar diversos métodos em comum, tal qual GetHashString, que retorna o valor Hash da string fornecida, e Update, que atualiza o valor Hash.

FixedInt e FixedUInt

A partir desta versão o Delphi passa a contar com dois novos tipos Integer, denominados FixedInt e FixedUInt. Por definição ambos são do tipo inteiro com um tamanho fixado em 32 bits invariável, o que garante uma melhor interoperabilidade no seu uso tanto em projetos para 32-bit quanto 64-bit.

Como diferencial, FixedInt se apresenta como um tipo assinado (signed), abrangendo valores que vão de -2147483648 a 2147483647. Em seu uso prático, para projetos de Target Platform Windows (32 e 64-bit), FixedInt é equivalente ao tipo LongInt, enquanto que para as outras plataformas de destino (que inclui Mac OS X, iOS e Android) sua equivalência se faz ao tipo Integer tradicional. Já FixedUInt é considerado um tipo não-assinado (unsigned), cujo intervalo compreende os valores de 0 a 4294967295. Em complemento, o tipo LongWord é seu equivalente para projetos para a plataforma Windows, na mesma medida que o tipo Cardinal é seu equivalente para as plataformas Mac OS X, iOS e Android.

LongInt e LongWord

Ainda com relação a tipos inteiro, o XE8 apresenta um diferencial para os tipos LongInt e LongWord, especificamente ao

Copyright - Proibido copiar ou distribuir. Todos os direitos reservados para DevMedia

desenvolvimento para a plataforma iOS 64-bit. Tal mudança se refere à duplicação do tamanho de bytes suportado pelo tipo, de 4 para 8 bytes, exclusivo em sua atuação nesta plataforma. Para todas as outras, incluindo iOS 32-bit e Windows 64-bit, o tamanho de 4 bytes é mantido.

IDE e Compilador

As novidades nessa versão do Delphi incluem também novos aspectos singulares que abrangem o próprio IDE, que é o ponto base de interação com o usuário desenvolvedor. Isso inclui também toda sua parte de compilador que desde as versões mais recentes do produto, apresenta uma gama de elemento que transcende a engine tradicional para Win32.

Nova Target Platform: iOS Device – 64-bit

Em seu cenário recente o Delphi acabou por incorporar a seu contexto uma gama de plataformas distintas, em complemento ao seu tradicional suporte ao Win32. Assim, a partir de uma mesma base de código, o desenvolvedor consegue agora produzir aplicativos tanto para Desktop quanto Mobile, em quatro opções diferentes de sistemas operacionais: Windows, Mac OS X, iOS e Android. Como já é sabido, na prática, o direcionamento de um projeto a uma plataforma alvo se dá pela configuração de sua Target Platform, disponível no Project Manager do IDE. Como novidade, a versão XE8 conta agora com o acréscimo de mais uma nova opção de plataforma de destino, denominada iOS Device – 64-bit, tal como mostra a (**Figura 7**). Sua nomeação autoexplicativa acaba por determinar seu fundamento, que é a produção de aplicativos para dispositivos iOS 64-bit.

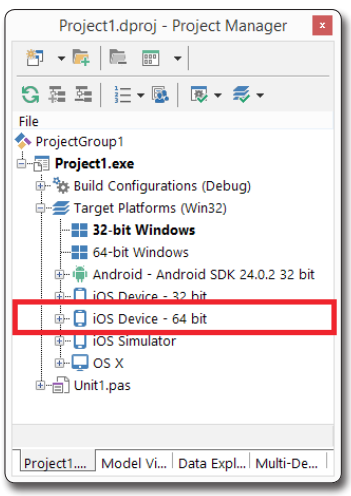


Figura 7. Nova Target Platform iOS Device – 64 bit

Novo compilador para iOS 64-bit

Em vista das recentes mudanças no cenário Apple relacionadas ao desenvolvimento de aplicativos para os seus dispositivos, em que agora a App Store exige que aplicativos para iOS 32-bit ofereçam suporte a dispositivos iOS 64-bit, nesta versão XE8 do Delphi são apresentadas melhorias a fim de atender a este quesito.

DCCIOSARM64 é o nome do compilador nativo que, a partir deste release, fica responsável pela produção de aplicações iOS tidas como universais – Universal iOS Apps – que refletem então aplicativos Delphi para a plataforma, prontas a rodar tanto em dispositivos 32 quanto 64-bit. No aspecto prático, este recurso é habilitado por meio da opção "Generate iOS universal binary file (armv7 + arm64)", disponível nas opções de projeto cuja Target Platform esteja marcada exclusivamente como iOS Device 64-bit.

Multi-Device Preview

Multi-Device Preview é o nome de um novo recurso que dá ao desenvolvedor uma prévia do visual de uma mesma aplicação, ainda em tempo de Design, em diversos aparelhos distintos, facilitando eventuais ajustes necessários. Em termos de localização, o recurso fica disponível no menu View (View > Multi-Device Preview) da ferramenta e é habilitado para projetos do tipo Multi-Device. Para exemplificar melhor o que foi dito, a **Figura 8** mostra então o Multi-Device Preview em ação, a partir de um projeto FireMonkey.

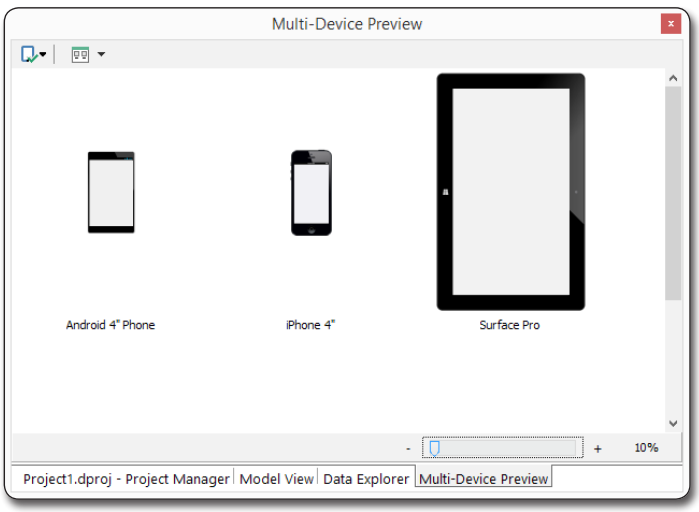


Figura 8. Multi-Device Preview

Em razão do próprio rumo que a ferramenta tomou em seus últimos tempos, este recurso acabou por se tornar uma necessidade natural e essencial ao desenvolvedor que a partir de uma base de código única, irá produzir aplicativos para diferentes plataformas e dispositivos. Até a versão anterior da ferramenta, este preview da aplicação em tempo de design para devices diferentes se dava por meio de uma opção única, referente a cada dispositivo, o que acabava por gerar uma ação repetitiva ao desenvolvedor. Agora com a chegada do Multi-Device Preview ganha-se em produtividade.

Device Manager

O Device Manager surge como um recurso essencial ao aspecto multi-device que a ferramenta tem tomado em tempos recentes. Por definição, ele se caracteriza como uma grade de device presets, que nada mais são do que predefinições visuais de dispositivos, conforme pode ser visto na **Figura 9**.

DEVMEDIA Copyright - Proibido copiar ou distribuir. Todos os direitos reservados para DevMedia

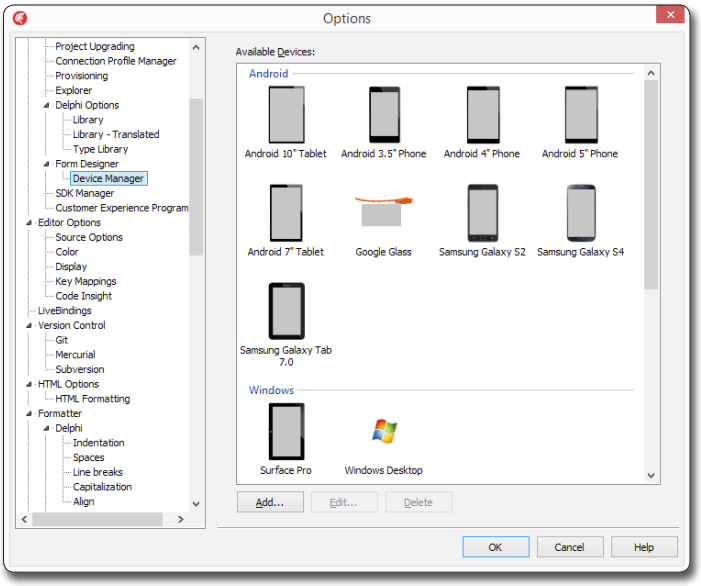


Figura 9. Device Manager

Logo, cada preset desse pode ser usado em tempo de design, como apoio à parte visual dos formulários de projetos construtivos de aplicações FireMonkey Multi-Device. Sem deixar de citar, por padrão, a própria instalação do Delphi XE8 já traz consigo uma série de device presets elaborados, prontos para uso no IDE.

Essas predefinições refletem o aspecto de alguns dispositivos genéricos disponíveis no mercado, tais como iPhone, Microsoft Surface e OS X, além de outros mais específicos, como Samsung Galaxy S4 e Samsung Galaxy Tab 7.0. Essas definições ficam então centralizadas em um arquivo especial denominado DevicePresets.xml (até o XE7 tratado como MobileDevices.xml), localizado em:

```
C:\Users\<nome_do_usuario>\AppData\Roaming\Embarcadero\BDS\16.0
```

Analisando seu conteúdo é possível notar a menção a algumas das predefinições citadas anteriormente, tal como o trecho a seguir da **Listagem 1**, referente ao device iPhone 4.

Em decorrência disso, fica fácil pressupor que o Device Manager provê também recursos suficientes para a manipulação de novos device presets, o que inclui a inclusão, alteração e exclusão destes. Essas ações ocorrem pelo acionamento dos botões indicativos – Add, Edit e Delete – presentes na própria estrutura do Device Manager, cujos resultados incidem diretamente sobre o conteúdo do arquivo especial citado, refletindo posteriormente também sobre o IDE. Em complemento, uma alternativa a este cenário se dá ainda pela manipulação direta do arquivo em questão. Em suma, a definição de um device preset envolve as seguintes informações relacionadas ao dispositivo:

- Apelido, um nome amigável de exibição no IDE;
- Orientações do dispositivo;
- Tamanho de exposição (display size) relacionada a cada orientação;
- Tamanho e posição da barra de status;
- Imagem para o frame do dispositivo.

Listagem 1. Predefinições para iPhone 4

```
...
<MobileDevice>
  <Displayname>iPhone 4"</Displayname>
  <Name>iPhone4in</Name>
  <DevicePlatform Default = "True">2</DevicePlatform>
  <FormFactor>2</FormFactor>
  <Portrait Enabled="True" Width="320" Height="568" Top="127" Left="39"
    StatusbarHeight="20" Artwork="$(BDS)\ObjRepos\en\Devices\images\
    iphone5.cpng" StatusBarPos="0"/>
  <UpsideDown Enabled="True" Width="320" Height="568" Top="124" Left="34"
    StatusbarHeight="20" Artwork="$(BDS)\ObjRepos\en\Devices\images\
    iphone5180.cpng" StatusBarPos="0"/>
  <LandscapeLeft Enabled="True" Width="568" Height="320" Top="39" Left="124"
    StatusbarHeight="20" Artwork="$(BDS)\ObjRepos\en\Devices\images\
    iphone590.cpng" StatusBarPos="0"/>
  <LandscapeRight Enabled="True" Width="568" Height="320" Top="34"
    Left="127" StatusbarHeight="20" Artwork="$(BDS)\ObjRepos\en\Devices\
    images\iphone5270.cpng" StatusBarPos="0"/>
</MobileDevice>
...
```

Citando a edição manual do arquivo DevicePresets.xml, tal atividade acabar por exigir o conhecimento adequado das tags de marcação envolvidas. Em vista disso, a seguir são expostas as tags tidas como essenciais:

- <MobileDevice>: tag principal que essencialmente define um dispositivo;
- <Name>: nome único do Device;
- <DisplayName>: nome "amigável" de exibição para o Device;
- <DevicePlatform>: plataforma relacionada ao Device, sendo 0 para Windows, 1 para Mac OS X, 2 para iOS e 3 para Android;
- <FormFactor>: definição da forma do dispositivo entre nove opções enumeradas, onde as principais são Desktop (0), Phone (2) e Tablet (3).
- <UserData>: determina se esta definição poderá ser editada a partir do Device Manager;
- <Portrait>, <UpsideDown>, <LandscapeLeft> e <LandscapeRight>: cada uma se refere à definição de uma orientação específica a ser usada pelo Device.

Add Featured Files

Este recurso toma a forma de uma janela de diálogo que é acionada toda vez que um novo arquivo com dependência é adicionado ao projeto através do Project Manager. Aqui, o Delphi estabelece "arquivos com dependência" como sendo aqueles com as seguintes extensões: .gdb, .ib, .fb, .fdb, .dbf, .db, .s2db, .s3db e .sdb. Conforme pode ser notado, em sua essência, tais extensões se referem a arquivos de dados, tal como .gdb, representativo de arquivos de banco de dados InterBase e .fdb, de arquivos de banco de dados Firebird. Em comum, todos eles implicam a inclusão de algum recurso complementar ao deployment configuration do projeto.

A fim de antecipar e facilitar todo este processo, o Delphi agora acaba por exibir a janela de Add Featured Files no exato momento após a inclusão de qualquer um desses arquivos ao projeto. De forma ilustrativa, a **Figura 10** esboça uma situação em que um

Copyright - Proibido copiar ou distribuir. Todos os direitos reservados para DevMedia

arquivo .gdb foi adicionado ao projeto, fazendo com que a janela Add Featured Files fosse acionada, já contemplando os arquivos complementares necessários para o ideal deploy futuro da aplicação.

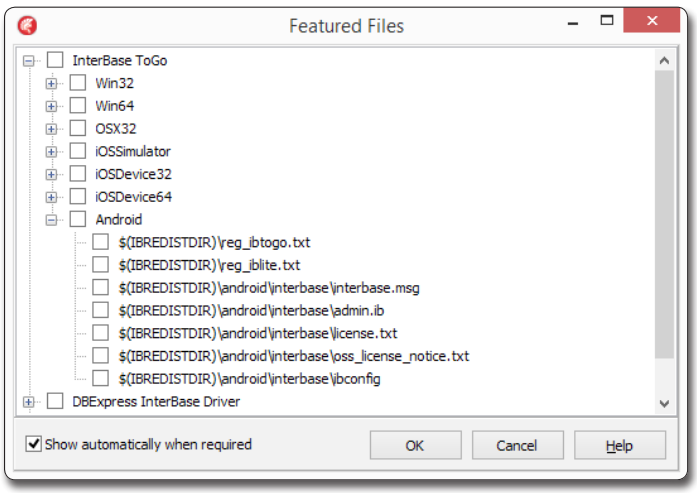


Figura 10. Janela Add Featured Files

Desabilitando Built-in RAD Studio Java Libraries

Outra novidade trazida especificamente para projetos Android, diz respeito ao descarte de bibliotecas Java built-in em um projeto. Neste contexto, o termo built-in é empregado para designar as Java Libraries "embutidas" nativamente pelo Delphi para seus projetos deste segmento. Por questões óbvias, de acordo com a essência do projeto, uma ou outra biblioteca acaba por se tornar desnecessária e sua remoção é quase que obrigatória para tornar a aplicação mais enxuta e otimizada. Todavia, até o XE7 o processo de remoção de uma Built-in Java Library de um projeto era algo não tão trivial, e fundamentalmente envolvia a manipulação de um arquivo complementar denominado classes.dex. A fim de simplificar esta abordagem, o Delphi XE8 possibilita que qualquer built-in Java library seja desabilitada por meio do próprio Project Manager, tal como mostra a **Figura 11**.

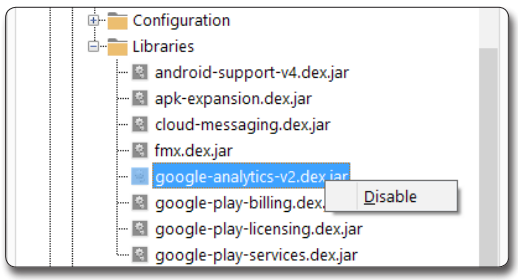


Figura 11. Opção Disable para Built-in Java Libraries

Melhorias para Git

A integração do IDE com o sistema de gerenciamento de código-fonte Git (**BOX 1**) é melhorado nesta versão por meio do acréscimo de duas novas funcionalidades:

• Suporte a autenticação para conexões a repositórios privados remotos – Authenticate;

• Suporte a envios de atualizações a repositórios remotos – Push;

Além disso, agora temos a atualização da cópia de trabalho (working copy) com as últimas alterações provindas do repositório remoto – Pull.

BOX 1. Git

Git é um sistema de controle de versão distribuído, de caráter livre, voltado ao gerenciamento de código-fonte. Sua concepção foi iniciada por Linus Torvalds, o "criador do Linux", sendo adotada posteriormente por diversos projetos. Em sua estrutura prática, cada diretório de trabalho do Git é considerado um repositório, o que implica o envolvimento de um histórico completo de log. Em tempos recentes, o suporte a Git foi adicionado ao Delphi, fazendo sua popularidade crescer ainda mais em meio à comunidade.

Suporte a Mercurial

Ainda relacionado a gerenciamento de código-fonte, o IDE do Delphi XE8 traz ainda como novidade o suporte a mais um sistema de controle de versão, agora o Mercurial, que se junta aos coirmãos anteriores, Subversion e Git. Em uma breve definição, salvo suas peculiaridades, o Mercurial tem sua essência aproximada ao Git, uma vez que se caracteriza como sendo um distributed version control system ou, em português, sistema de controle de versão distribuído. Neste início de parceria (Delphi-Mercurial), três são as ações principais quanto ao uso da ferramenta sob o contexto do XE8: Clone, Commit e Show Log. A primeira efetua uma cópia íntegra de um repositório remoto para o início dos trabalhos locais. Já a segunda, Commit, envia as alterações para o repositório local, enquanto a última simplesmente mostra as informações de log do projeto.

A partir disso, sem sombra de dúvidas, este recurso é digno de um artigo completo sobre o mesmo, uma vez que envolve nuances que o fomentam como um produto bastante singular.

Settings Migration Tool

Settings Migration Tool é o nome de uma ferramenta externa que acompanha a instalação do Delphi XE8, cuja função é realizar a importação e exportação de definições de configuração (configuration settings) entre versões distintas do IDE. De forma fundamental, seu principal uso se dá pela eventual migração/transporte de definições de configuração do IDE a diferentes instâncias instaladas em ambientes distintos. Ainda dada sua utilização, dois aspectos restritivos são impostos pela ferramenta. A primeira se reflete em seu suporte, previsto a todas as versões do produto a partir do Delphi 7, logo, outros releases clássicos, tal como o Delphi 5, estão fora do contexto proposto. Outra restrição existente se dá pelo próprio processo de migração, que ocorre sempre em linha linear crescente, o que estabelece uma atividade entre versões de mesma numeração ou maior (Ex: de XE2 para XE8, nunca de XE2 para XE8).

Por padrão, o Settings Migration Tool é disponibilizado em:

C:\Program Files (x86)\Embarcadero\Studio\16.0\bin

A seguir, a **Figura 12** mostra sua janela de execução, a qual apresenta suas principais operações:

DEVMEDIA Copyright - Proibido copiar ou distribuir. Todos os direitos reservados para DevMedia

• Export settings to a migration file: exporta as definições de configuração para um arquivo XML tradicional, porém com uma extensão própria (.idesettings);
• Migrate settings to a newer product version: migra as definições de configuração para uma nova instância do IDE instalada na máquina corrente;
• Restore settings from backup: restaura as configurações a partir de uma cópia de segurança provida;
• Import settings from a migration file: importa um arquivo .idesettings.

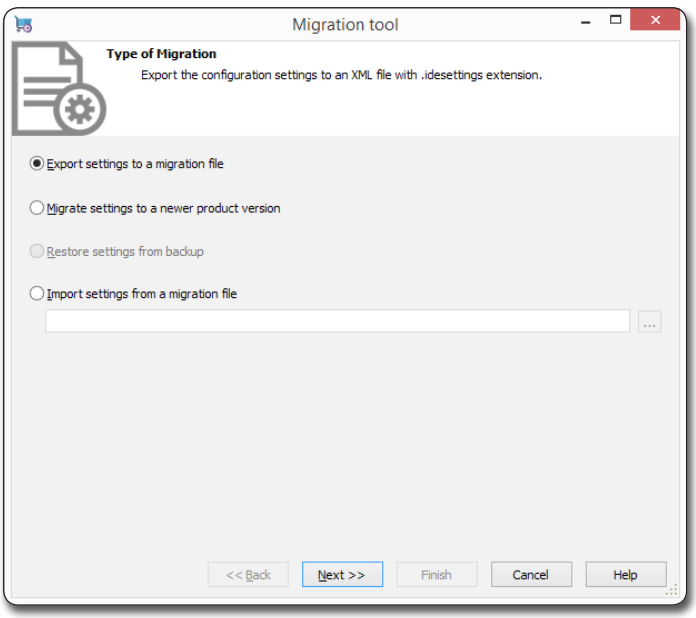


Figura 12. Setting Migration Tool

Castalia

O IDE do XE8 conta agora com a integração nativa a uma ferramenta já bastante conhecida em meio à comunidade de desenvolvedores Delphi, o Castalia. De forma breve, este plug-in pode ser definido como um facilitador de tarefas ao desenvolvedor, relacionadas ao desenvolvimento de software dentro do IDE. A partir de um ponto de vista mais conceitual, todas essas tarefas podem então ser subdivididas em quatro segmentos independentes:
• Recursos Visuais: compreende o conjunto de elementos visuais de apoio que se incorporam ao editor de código (Code Editor) do Delphi;
• Recursos Não-Visuais: compreende o conjunto de atalhos, os quais irão prover facilidades e funcionalidades durante a codificação;
• Controles de edição de código: faz referência aos controles incorporados ao IDE que colaboram na navegação de código;
• Menu: envolve as funcionalidades e estatísticas de código.

Tais quais outros recursos apresentados nesta versão do IDE, o Castalia mostra-se complexo o suficiente para o natural surgimento de artigos exclusivos sobre o tema, fundamentalmente direcionados ao seu uso prático.

Customer Experience Program

Customer Experience Program é a denominação dada a um programa colaborativo proposto pela Embarcadero, cujo intuito é coletar dados anônimos dos usuários da ferramenta durante sua interação com o IDE. Já existente há algum tempo em grande parte de seus concorrentes, esse tipo de programa visa a tomada de decisões mais assertivas, por parte da empresa fabricante, no que diz respeito a eventuais melhorias ou mudanças na experiência do usuário com o ambiente de desenvolvimento. Sob a ótica da empresa, tal usuário representa desde o profissional da área, passando por programadores casuais, até simples aventureiros.

Por envolver a transmissão de dados pessoais, a Embarcadero apenas disponibiliza o acesso ao programa no próprio IDE do Delphi XE8, cabendo ao usuário seu ingresso voluntário ao Customer Experience Program. Logo, tal opção é acessível por meio menu Tools > Options > Environment Options, conforme mostra a **Figura 13**.

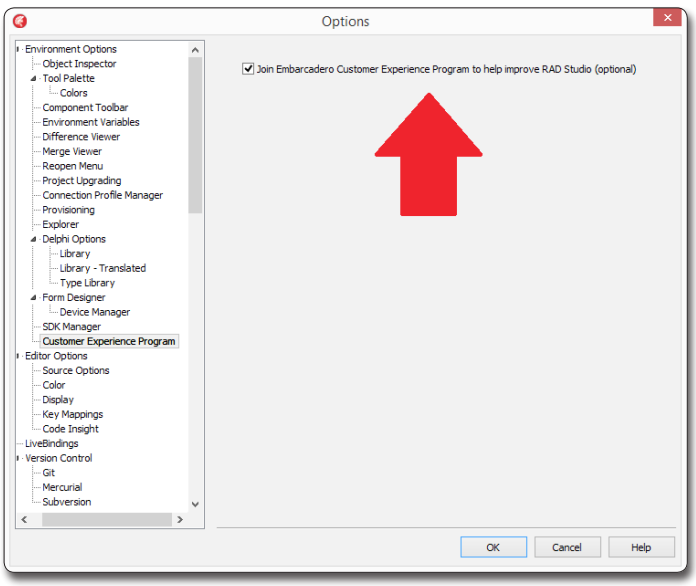


Figura 13. Opção Customer Experience Program

Novo sistema de Help off-line

O arquivo de Ajuda (Help) off-line que acompanha a instalação do Delphi XE8 apresenta um novo formato, se comparado com o formato apresentado até o XE7. Anteriormente o formato utilizado era denominado H2, e sua visualização ficava atrelada a um determinado visualizador. Agora o Microsoft Compiled HTML Help (.chm) é o novo formato adotado, cuja popularidade e simplicidade se mostram superiores à opção anteriormente existente.

Novidades para FireDAC

No cenário Delphi recente, o FireDAC se consolidou definitivamente como a principal opção nativa para acesso a dados, numa substituição quase que completa do dbExpress. Em vista disso, a cada nova versão da ferramenta a expectativa (e necessidade) por novidades e melhorias torna-se bastante grande.

Copyright - Proibido copiar ou distribuir. Todos os direitos reservados para DevMedia

No XE8 o principal atrativo do FireDAC é o seu novo driver nativo, baseado em ODBC, para Teradata Database (**BOX 2**). Seguindo por esta linha, outra melhoria a ser citada se dá pela atualização de seu driver para SQLite, elemento este fundamental para cenários Mobile nos dias atuais.

BOX 2. Teradata Database

Teradata Database é a denominação dada a um software de banco de dados que se responsabiliza por armazenar as informações direcionadas a ele em um ambiente de arquitetura paralela aberta. Funcionalmente, o Teradata Database é executado sob uma plataforma exclusiva nomeada Teradata Active Enterprise Data Warehouse. Ambas as tecnologias são parte integrante do conjunto de soluções desenvolvido pela empresa Teradata Corporation.

Novidades para EMS

O Enterprise Mobility Services, ou simplesmente EMS, é a solução pronta para aplicações distribuídas que a Embarcadero incorporou ao Delphi em sua versão anterior (XE7). Dadas as suas pretensões era natural de se prever uma evolução bastante consistente do EMS em seus releases posteriores ao seu lançamento. Em vista disso, agora no XE8 o framework apresenta novos acréscimos e melhorias pontuais, a começar pela adição de um novo resource, nomeado EMS Installations, que fica responsável, entre outras coisas, por mostrar os dados do EMS Console Server. Ainda relacionado aos dados do Server, toda sua parte analítica pode ser agora exportada para arquivos de extensões .csv.

Além de EMS Installations, outro novo resource incorporado ao contexto é o EMS Push Resource, que fundamentalmente armazena os dados de mensagens de push enviadas do EMS Server aos dispositivos registrados. Outra novidade apresenta-se sob o nome de EMS Management Console Application (EMSManagementConsole.exe), que nada mais é do que uma aplicação pronta de amostra, distribuída junto da instalação do Delphi XE8, que pode ser usada para aspectos básicos do contexto EMS, tais como manipulação de dados do servidor e envio de mensagens a dispositivos cliente.

O aplicativo é fisicamente disponibilizado em duas versões, sendo uma 32 e outra 64 bits, localizadas nativamente em:

```
C:\Program Files (x86)\Embarcadero\Studio\16.0\bin
C:\Program Files (x86)\Embarcadero\Studio\16.0\bin64
```

Além disso, dentre as novidades podemos destacar os aperfeiçoamentos na integração com FireDAC, o novo componente EMSClientAPI para simplificar o desenvolvimento do lado do cliente e as análises aperfeiçoadas para usuários e grupos.

Novidades de Third-Party

Além de novos recursos próprios, o Delphi XE8 conta ainda com novidades de terceiros, os chamados elementos third-party, que se agregam ao produto em meio a sua instalação padrão.

Box2D

Box2D é o nome de uma engine de código aberto relacionada à física 2D direcionada à criação de jogos. Sua utilização em projetos Delphi se dá pela simples configuração do Search Path do mesmo, disponível nas opções de compilador. Neste ponto, o direcionamento deve ser feito para "${BDS)\source\FlatBox2D". Isso é o suficiente para que as units sejam encontradas pelo IDE e referenciadas no código, habilitando o uso das classes e recursos do Box2D.

DUnitX

O novo Delphi conta agora também com suporte a um novo framework de testes, o DUnitX que, conforme sua nomeação sugere, pode ser interpretado como uma evolução do já tradicional DUnit. Com uma gama de métodos a asserções próprias, a utilização do DUnitX é prevista para projetos Desktop, o que atualmente inclui Windows (32 e 64 bits) e Mac OS X.

GetIt

Por meio de seu menu Tools, o novo IDE do Delphi provê acesso a um novo utilitário denominado GetIt que, fundamentalmente, permite o download e instalação de pacotes exclusivos, tratados aqui como "GetIt Packages". Cada package pode conter então diversos recursos de apoio ao contexto do desenvolvimento, tais como bibliotecas, novos componentes e extensões da ferramenta.

Dentre as novidades do XE8 os destaques ficam por conta de quatro pilares: o suporte a Mercurial, a integração nativa com o Castalia, as melhorias no Enterprise Mobility Services e a adição do DUnitX.

Autor

Fabrício Hissao Kawata
fabricio.kawata@bol.com.br
Formado em Processamento de Dados pela FATEC-TQ e pós-graduado em Engenharia de Componentes. Atua como Analista Programador Delphi há 9 anos.

Links:

RAD Studio XE8 - Página oficial do produto
http://www.embarcadero.com/products/rad-studio

RAD Studio XE8 – Download Trial
https://downloads.embarcadero.com/free/rad_studio

Você gostou deste artigo?

Dê seu voto em ***www.devmedia.com.br/clubedelphi/feedback***

Ajude-nos a manter a qualidade da revista!

DEVMEDIA Copyright - Proibido copiar ou distribuir. Todos os direitos reservados para DevMedia

Desenvolvendo um Sistema Financeiro em Delphi– Parte 3

Implementação da camada de acesso a dados

Este artigo faz parte de um curso

Fique por Dentro

Neste artigo iremos tratar de um tópico muito importante em um projeto orientado a objetos, que é a camada de acesso a dados. É mostrado como implementar classes DAO, onde temos o mapeamento das classes de negócio para tabelas no banco de dados e o mapeamento das propriedades para as colunas das tabelas criadas, e vice-versa. Veremos objetos sendo transformados em linhas na tabela e propriedades transformadas em colunas.

O Delphi, no que diz respeito a linguagem, implementa todos os requisitos de uma linguagem orientada a objetos, pois nos permite fazer abstração, encapsulamento, herança e polimorfismo. Porém, o que mais vemos nos desenvolvedores Delphi é o desenvolvimento de aplicações de forma estruturada, onde são explorados apenas os recursos RAD que o IDE proporciona.

Um dos grandes pontos negativos de trabalhar com Delphi é a falta de um framework de mapeamento objeto-relacional robusto e que seja apoiado pela Embarcadero. Existem iniciativas como o DORM (open-source) e o Aurelius (comercial), mas são soluções que ainda não se tornaram populares. Neste artigo veremos uma solução manual para resolver esse problema de mapeamento objeto-relacional. Existem boas práticas, como a Inversão de Controle (**BOX 1**), que no Delphi não possui um framework apoiado e incentivado pela Embarcadero.

Apesar da ferramenta não oferecer um framework nativo para trabalharmos com orientado a objetos, a Delphi Language possui todos os recursos necessários para desenvolvermos orientado a objetos, como Generics e Métodos Anônimos.

Impedância Objeto Relacional

A Orientação a Objetos traz como principal vantagem sobre a estruturada a representação de objetos de maneira bastante semelhante ao mundo real, o que torna-se um desafio a mais quando precisamos fazer a persistência destes nos bancos de dados relacionais.

BOX 1. Inversão de Controle e Injeção de Dependência

Este princípio é a base para qualquer bom design de software orientado a objetos. O princípio da Inversão de Dependência nos diz que módulos de alto nível não devem ser dependentes de módulos de baixo nível, ambos devem depender de abstrações, que por sua vez, não devem depender de detalhes.

Inverter a dependência faz com que o cliente não fique frágil a mudanças relacionadas a detalhes de implementação, isto é, mudar um detalhe da implementação não faz com que sejam necessárias alterações no cliente. Este princípio é bastante presente em muitos padrões de projeto, pois a maioria deles definem uma interface para que não haja dependências de implementações.

Outro padrão que geralmente anda junto com o princípio de inversão de dependência é a Injeção de Dependências, que é uma forma de conseguir a inversão de controle. Nesta solução, as dependências entre os módulos não são definidas programaticamente, mas sim pela configuração de uma infraestrutura de software (container) que é responsável por injetar em cada classe suas dependências declaradas.

O padrão de Injeção de dependências sugere que uma conexão com banco de dados seja injetada na classe. Com isso, além de inverter a dependência, a classe não precisa se preocupar com o ciclo de vida das suas dependências (no exemplo da conexão, a classe não precisa abrir ou fechar conexão, apenas receber uma referência desta conexão e a utiliza).

Copyright - Proibido copiar ou distribuir. Todos os direitos reservados para DevMedia

Existem soluções de bancos de dados orientados a objetos, mas estes ainda não se tornaram populares, sendo os bancos de dados relacionais ainda muito mais utilizado que estes.

Não existe uma conversão direta entre os objetos que construímos nas linguagens de programação e o banco de dados relacionais, por isso é necessário criarmos o que é chamado de Mapeamento Objeto-Relacional, onde objetos são transformados geralmente em registros em uma tabela e vice-versa.

Mapeamento Objeto Relacional

Isso surgiu porque precisamos salvar o estado de um objeto (atributos, herança, polimorfismo) em tabelas do banco de dados. O mapeamento básico pode ser feito através de conversões manuais ou através de frameworks de persistência, onde ambas possuem vantagens e desvantagens, mas a conversão manual nos dá uma melhor performance, enquanto que com frameworks de persistência não precisamos nos preocupar com escrita de SQL.

Em Delphi os frameworks de mapeamento objeto relacional geralmente nos possibilitam a configuração de persistência através de arquivos XML ou o recurso de annotations ou custom atributes.

No mapeamento objeto relacional básico o que temos é o mapeamento de classes para uma tabela do banco de dados, temos também o mapeamento de objetos da classe para registros da tabela e as propriedades da classe para colunas da tabela.

Padrão DAO (Data Access Object)

O padrão de projeto DAO surgiu com a necessidade de separarmos a lógica de negócios da lógica de persistência de dados.

Este padrão permite que possamos mudar a forma de persistência sem que isso influencie em nada na lógica de negócio, além de tornar nossas classes mais legíveis.

Classes DAO são responsáveis por trocar informações com o SGBD e fornecer operações CRUD e de pesquisas, além de ser capaz de buscar dados no banco e transformar em objetos ou lista desses através de listas genéricas. Também deverá receber os objetos, converter em instruções SQL e mandar para o bando de dados. Toda interação com a base se dará através destas classes, nunca das classes de negócio, muito menos de formulários.

Se aplicarmos este padrão corretamente será abstraído completamente o modo de busca e gravação dos dados, tornando isso transparente para aplicação, facilitando muito na hora de fazermos manutenção na aplicação ou migração de banco de dados.

Também conseguimos centralizar a troca de dados com o SGBD, assim teremos um ponto único de acesso a dados e nossa aplicação um ótimo design orientado a objeto.

Generics

Generics é um recurso que foi incorporado na versão 2009 do Delphi, que permite criar estruturas genéricas. Na orientação a objetos podemos definir o tipo de retorno de uma lista de objetos e somente serão permitidas inserções de objetos daquele tipo, do contrário ocorrerá um erro ainda em tempo de compilação, assim, evitando erros em tempo de execução do tipo Invalid Type Casting, muito comum para quem trabalhava com listas de objetos em versões anteriores a 2009 em Delphi. Para fazer uso de Generics temos que declarar o namespace System.Generics.Collections na seção uses do código.

As duas principais classes deste namespace são a TList<T> e TobjectList<T>. A Classe TList<T> possibilita guardar coleções de qualquer tipo de dados, tanto primitivos quanto objetos. Já a classe TObjectList<T>, que é uma especialização de TList, pode armazenar somente objetos. Ela tem a vantagem de que no momento que liberarmos sua instância da memória, todos os objetos contidos nela também são liberados, diferentemente das instâncias de TList, onde no caso de as usarmos para armazenar objetos, precisamos liberar cada objeto individualmente da memória.

FireDAC

FireDAC é a versão da Embarcadero para o AnyDAC, que foi adquirida e integrada ao Delphi e o C++ Builder. No Delphi o FireDAC está disponível na versão XE3 e XE4 com um conjunto de componentes extras, e a partir da versão XE5 vem na instalação padrão.

É um conjunto de componentes de alta performance, de fácil utilização e provê conexão com vários bancos de dados, tanto locais quanto coorporativos. Cada banco de dados possui um driver e trabalhado de forma diferente. Os drivers do FireDAC são nativos para cada banco de dados e ainda possui pontes para ODBC e dbExpress. Vários bancos de dados são suportados, entre eles temos: MySQL, SQL Server, Oracle e SQLite.

Para quem tem projetos multicamadas com Delphi e DataSnap (**BOX 2**), ou também servidores REST, basta migrar a parte Server de DBExpress para FireDAC, pois o TFDQuery é um TDataSet, portanto compatível com o TDataSetProvider e TClientDataSet. Na parte Client, continua o TClientDataSet sem necessidade de qualquer alteração. Tem suporte a Firemonkey e VCL e possui um conjunto de componentes visíveis e não-visíveis, DataSets, Adapters.

BOX 2. DataSnap

O Datasnap é o mecanismo desenvolvimento multicamadas do Delphi. Servidores de aplicação Datasnap podem ser desenvolvidos tanto em Delphi quanto C++ Builder, porém a grande vantagem está no cliente, que pode ser desenvolvido em qualquer linguagem de programação que tenha suporte a JSON. Em 2009 foi criado um novo driver para o DBExpress, mas ao invés deste driver conectar-se a um banco de dados qualquer, ele se conectaria a um servidor de aplicação. Assim, a partir do Delphi 2009 a conexão do cliente com o servidor de aplicação passou a ser feita utilizando DBExpress trafegando os dados utilizando o protocolo TCP/IP.

Classe Base DAO

Abra o projeto no Delphi para continuarmos. Para facilitar o trabalho com as classes DAO, criamos uma classe base a qual será herdada por todas as classes DAO que irão interagir com o banco de dados, conforme pode ser visto na **Listagem 1**.

DEVMEDIA Copyright - Proibido copiar ou distribuir. Todos os direitos reservados para DevMedia

Listagem 1. Classe base TDAO

```
01 unit DAO.Base;
02 interface
03 uses
04  DAO.ConnectionFactory, FireDAC.Comp.Client;
05 type
06  TDAO = class
07  protected
08   Connection: TFDConnection;
09   function GetKeyValue(ATable: string): Integer;
10  public
11   constructor Create;
12   destructor Destroy; override;
13  end;
14 implementation
15 {TDAO}
16 constructor TDAO.Create;
17 begin
18  Connection := TConnectionFactory.GetConnection;
19 end;
20 destructor TDAO.Destroy;
21 begin
22  Connection.Free;
23  inherited;
24 end;
25 function TDAO.GetKeyValue(ATable, AColumn: string): Integer;
26 var
27  SQL: string;
28  Id: Integer;
29 begin
30  SQL := 'select coalesce(max(' + AColumn + '),0) + 1 from ' + ATable;
31  Id := Integer(Connection.ExecSQLScalar(SQL));
32  result := Id;
33 end;
34 end.
```

Nesta classe temos a declaração de uma referência para TFDConnection com escopo protegido, de maneira que possamos acessar este atributo em todas as classes derivadas desta. Usaremos este objeto em todas as classes descendentes, tanto para executar comandos de consulta quanto comandos de atualização no banco de dados.

Temos também o método GetKeyValue, que deverá ser chamado toda vez que precisarmos gerar uma nova chave para alguma tabela, bastando passar por parâmetro o nome e a coluna que desejamos receber a chave.

Utilizamos o método ExecSQLScalar do FireDAC para buscarmos o valor desejado. Este deve ser usado sempre que nossa consulta retornar apenas uma informação. O seu retorno é do tipo variant, por isso fizemos um typecast antes de fazermos atribuição à variável Id.

O mecanismo de geração da chave que escolhemos foi o de pegar o máximo valor existente na coluna da tabela mais um, porém poderíamos usar qualquer outro mecanismo, como o uso de Generators do Firebird, por exemplo.

Toda comunicação com o banco de dados será feita através de um objeto da classe TFDConnection que será disponibilizado pela classe TConnectionFactory, como mostra a **Listagem 2**.

A classe TConnectionFactory é uma fábrica de conexões, ou seja, cada vez que precisarmos de uma conexão com o FireDAC iremos recorrer a ela, que nos dará uma conexão configurada e pronta para executarmos comandos de consulta e atualização no banco de dados Firebird.

Neste exemplo, fixamos os parâmetros de conexão no atributo ConnectionString, mas o ideal é que estes venham num arquivo separado como, por exemplo, um arquivo INI ou XML, para que possamos mudar as informações do banco de dados sem que seja necessário compilar o aplicativo.

Listagem 2. Relembrando a classe TConnectionFactory

```
01 unit DAO.ConnectionFactory;
02 interface
03 uses
04  FireDAC.Comp.Client, FireDAC.Stan.Def, FireDAC.Stan.Error,
    FireDAC.Stan.Async,
05  FireDAC.VCLUI.Wait, FireDAC.DApt, FireDAC.Phys.FB;
06 type
07  TConnectionFactory = class
08  private
09   class var FDConnection: TFDConnection;
10  public
11   class function GetConnection: TFDConnection;
12  end;
13 implementation
14 {TConnectionFactory}
15 class function TConnectionFactory.GetConnection: TFDConnection;
16 begin
17  FDConnection := TFDConnection.Create(nil);
18  FDConnection.ConnectionString := 'DriverID=FB;Server=127.0.0.1;
    Database=D:Database\DBFINANCEIRO.FDB;User_name=SYSDBA;
    Password=masterkey';
19  FDConnection.Connected := True;
20  Result := FDConnection;
21 end;
22 end.
```

Classe TOrigemDAO

Para a classe de modelo TOrigem criaremos sua correspondente DAO, chamada TOrigemDAO, apresentada na **Listagem 3**. Ela deve ser capaz de receber um objeto do tipo TOrigem e persistilo na tabela ORIGENS do banco de dados. Da mesma forma, a classe TOrigemDAO deve ser capaz de buscar na base de dados um registro da tabela ORIGENS e convertê-lo em um objeto da classe TOrigem, como mostra a **Listagem 4**.

Para facilitar a criação de objetos da classe TOrigem, criamos um construtor que recebe todos atributos da classe, de maneira que, ao criarmos uma instância de TOrigem, já populamos todas as suas propriedades, nos poupando diversas linhas de código, como podemos ver nas linhas 25 a 30 da **Listagem 4**.

Na **Listagem 5** temos todos os métodos para fazer as operações básicas da classe TOrigemDAO. No método insert não recebemos todos os campos da tabela por parâmetro e sim um objeto da classe TOrigem, que virá com todas as propriedades populadas, ou de uma aplicação de teste ou da interface com o usuário.

Nos métodos Insert, Update e Delete temos a declaração de uma variável SQL do tipo String, onde atribuiremos o comando SQL. Especificamente nos métodos Insert e Update faz-se necessária a criação de uma variável que representa o TipoOrigem, do tipo Char.

Copyright - Proibido copiar ou distribuir. Todos os direitos reservados para DevMedia

Listagem 3. Interface da Classe TOrigemDAO

```
01 TOrigemDAO = class(TDAO)
02 public
03   function Insert(AOrigem: TOrigem): Integer;
04   procedure Update(AOrigem: TOrigem);
05   procedure Delete(AId: Integer);
06   function FindById(AId: Integer): TOrigem;
07   function FindAll: TObjectList<TOrigem>;
08 end;
```

Listagem 4. Classe TOrigem

```
01 unit Model.Origem;
02
03 interface
04
05 type
06  TTipoOrigem = (toPagamento, toRecebimento);
07
08  TOrigem = class(TObject)
09  private
10    FId: Integer;
11    FDescricao: string;
12    FTipoOrigem: TTipoOrigem;
13  public
14    property Id: Integer read FId write FId;
15    property Descricao: string read FDescricao write FDescricao;
16    property TipoOrigem: TTipoOrigem read FTipoOrigem write FTipoOrigem;
17    constructor Create(AId: Integer; ADescricao: string;
      ATipoOrigem: TTipoOrigem); overload;
18    constructor Create; overload;
19  end;
20
21 implementation
22
23 { TOrigem }
24
25 constructor TOrigem.Create(AId: Integer; ADescricao: string;
    ATipoOrigem: TTipoOrigem);
26 begin
27   FId := AId;
28   FDescricao := ADescricao;
29   FTipoOrigem := ATipoOrigem;
30 end;
31
32 constructor TOrigem.Create;
33 begin
34 end;
35
36 end.
```

Isso é necessário, pois quando recebemos o objeto Aorigem por parâmetro, este possui internamente uma propriedade do tipo enumeração, onde é armazenado o tipo de origem, se é toPagamento ou toRecebimento. Naturalmente essas não são as esperadas pela coluna TIPO_ORIGEM da tabela ORIGENS da base de dados, portanto necessita-se criar uma variável local que faz a conversão entre os dois tipos de dados antes de fazer a atualização no banco de dados.

No método Insert temos ainda a chamada do método GetKeyValue da classe base TDAO, onde passamos a tabela ORIGENS e o campo chave ID_ORIGEM, que será utilizada na hora de montarmos o SQL de inserção.

A string do Insert possuirá três parâmetros, um para cada coluna da tabela. Por último, chamamos o método ExecSQL do objeto Connection. Este método possui três parâmetros: o primeiro é o SQL que desejamos que executar, o segundo é um vetor com os parâmetros do SQL, e por último um vetor com os tipos de dados dos parâmetros passados. Este último parâmetro é opcional, e caso não seja informado, o próprio FireDAC se encarrega de identificar de qual tipo é. Nesse caso, achamos interessante já passar esta informação para que seja executado o mais performático possível.

Listagem 5. Métodos de Atualização (Insert, Update, Delete)

```
01 function TOrigemDAO.Insert(AOrigem: TOrigem): Integer;
02 var
03   SQL: string;
04   IdOrigem: Integer;
05   TipoOrigem: Char;
06 begin
07   case AOrigem.TipoOrigem of
08     toPagamento: TipoOrigem := 'P';
09     toRecebimento: TipoOrigem := 'R';
10   end;
11   IdOrigem := GetKeyValue('ORIGENS', 'ID_ORIGEM');
12   SQL := 'insert into ORIGENS values (:PAR1, :PAR2, :PAR3)';
13   Connection.ExecSQL(SQL, [AOrigem.Id, AOrigem.Descricao, TipoOrigem],
14     [ftInteger, ftString, ftString]);
15 end;
16
17 procedure TOrigemDAO.Update(AOrigem: TOrigem);
18 var
19   SQL: string;
20   TipoOrigem: Char;
21 begin
22   case AOrigem.TipoOrigem of
23     toPagamento: TipoOrigem := 'P';
24     toRecebimento: TipoOrigem := 'R';
25   end;
26   SQL := 'update ORIGENS set DESCRICAO = :PAR1, TIPO_ORIGEM = :PAR2 where
ID_ORIGEM = :PAR3';
27   Connection.ExecSQL(SQL, [AOrigem.Descricao, TipoOrigem, AOrigem.Id],
28     [ftString, ftString, ftInteger]);
29 end;
30
31 procedure TOrigemDAO.Delete(AId: Integer);
32 var
33   SQL: string;
34 begin
35   SQL := 'delete from ORIGENS where ID_ORIGEM = ' + IntToStr(AId);
36   Connection.ExecSQL(SQL);
37 end;
```

O método Update é bastante semelhante ao método Insert, só não precisamos fazer a geração de chave e mudamos a ordem de passagem dos parâmetros, já que o identificador desta vez é utilizado na cláusula where do SQL.

O método Delete é o mais simples de todos, pois somente recebemos por parâmetro o identificador do registro que deseja-se excluir do banco de dados e montamos o SQL com um único parâmetro, que é passado e executado a seguir. Desta vez, optamos, por uma questão de simplicidade, apenas concatenar o valor do AId após ele ser convertido para string.

Os métodos de busca de informações no banco de dados, presentes na **Listagem 6**, e criação de objetos são mais trabalhosos porque precisamos recuperar as linhas e convertê-las em objetos.

DEVMEDIA Copyright - Proibido copiar ou distribuir. Todos os direitos reservados para DevMedia

Para isso, usaremos outro objeto da arquitetura do FireDAC que é o FDQuery. Com ele podemos executar consultas na base de dados que retornam um DataSet que pode ser percorrido, e assim criarmos nossos objetos da classe TOrigem.

Listagem 6. Métodos de Consulta

```
01 function TOrigemDAO.FindAll: TObjectList<TOrigem>;
02 var
03  FDQuery: TFDQuery;
04  TipoOrigem: TTipoOrigem;
05  Origens: TObjectList<TOrigem>;
06 begin
07  FDQuery := TFDQuery.Create(nil);
08  try
09   FDQuery.Connection := Connection;
10   FDQuery.SQL.Text := 'select * from ORIGENS';
11   FDQuery.Open();
12   if FDQuery.FieldByName('TIPO_ORIGEM').AsString = 'P' then
13    TipoOrigem := toPagamento
14   else if FDQuery.FieldByName('TIPO_ORIGEM').AsString = 'R' then
15    TipoOrigem := toRecebimento;
16   Origens := TObjectList<TOrigem>.Create();
17   while not FDQuery.Eof do
18   begin
19    Origens.Add(TOrigem.Create(FDQuery.FieldByName('ID_ORIGEM').AsInteger,
20     FDQuery.FieldByName('DESCRICAO').AsString, TipoOrigem));
21   end;
22  finally
23   FDQuery.Free;
24  end;
25  result := Origens;
26 end;
27
28 function TOrigemDAO.FindById(AId: Integer): TOrigem;
29 var
30  Origem: TOrigem;
31  FDQuery: TFDQuery;
32  TipoOrigem: string;
33 begin
34  FDQuery := TFDQuery.Create(nil);
35  try
36   FDQuery.Connection := Connection;
37   FDQuery.SQL.Text := 'select * from ORIGENS
     where ID_ORIGEM = ' + IntToStr(AId);
38   FDQuery.Open();
39   if FDQuery.RecordCount = 0 then
40    raise Exception.Create('Objeto não encontrado na base de dados!');
41   Origem := TOrigem.Create();
42   Origem.Id := FDQuery.FieldByName('ID_ORIGEM').AsInteger;
43   Origem.Descricao := FDQuery.FieldByName('DESCRICAO').AsString;
44   TipoOrigem := FDQuery.FieldByName('TIPO_ORIGEM').AsString;
45   if TipoOrigem = 'P' then
46    Origem.TipoOrigem := toPagamento
47   else if TipoOrigem = 'R' then
48    Origem.TipoOrigem := toRecebimento;
49  finally
50   FDQuery.Free;
51  end;
52  Result := Origem;
53 end;
```

No método FindById buscamos um registro na base de dados a partir de um Id passado por parâmetro. Para executarmos um comando SQL no FDQuery criamos um objeto da classe TFDQuery dentro de um bloco try/except, para garantirmos que o objeto será liberado da memória no final de seu uso. O objeto FDQuery precisa de uma conexão para executar o comando, por esse motivo atribuímos a conexão na propriedade Connection o SQL e damos um Open na query.

Na sequência verificamos se a query retornou algum registro: caso não tenha retornado, lançamos uma exceção informando que nenhum registro foi encontrado na base de dados. Caso seja encontrado o registro, criamos um objeto da classe TOrigem e populamos suas propriedades com as colunas retornadas no DataSet do FDQuery.

No método FindAll é onde teremos a recuperação de todas as linhas da tabela ORIGENS em uma lista genérica de TOrigem. De início, dá para notar que o retorno deste método é um TObjectList do tipo TOrigem.

Para construirmos nossa lista criamos uma lista genérica de TOrigem e executamos uma query que retorna todas as linhas da tabela. Em seguida, percorremos todas as linhas do DataSet gerado e, para cada linha, criamos um objeto de TOrigem e adicionamos na lista. No final, apenas atribuímos a variável local Origens no retorno do método.

Classes TClienteDAO e TFornecedorDAO

Para facilitar o trabalho com os objetos TCliente e TFornecedor da **Listagem 7**, criamos construtores que recebem as propriedades por parâmetro e já as populam, facilitando bastante a criação dos objetos e adição nas listas.

A interface da classe TClienteDAO é bastante semelhante à classe TOrigemDAO, conforme podemos visualizar na **Listagem 8**. A diferença se dá pela adição de um novo método de busca chamado FindByName (linha 06), que servirá para realizarmos consultas no cadastro de clientes pelo nome.

Nesta classe não temos o método FindAll, utilizado para retornar todos os registros da tabela. Nesse caso, não devemos fazer este tipo de busca no banco de dados, pois com o aumento do tamanho da base de dados, pode-se retornar muitos registros e deixar lento o aplicativo.

Os métodos Insert, Update e Delete da classe TClienteDAO seguem a mesma linha do que já havíamos codificado na classe TOrigemDAO, conforme a **Listagem 9.** Veja que não temos nenhum atributo do tipo enumeração e não precisamos fazer nenhuma conversão, tornando ainda mais simples a codificação.

O método FindByName irá receber uma String com um nome do cliente ou parte do nome, e que será usada em conjunto com o operador Like, que permite fazermos pesquisa na base de dados. Utilizamos também a função QuotedStr, que irá adicionar aspas antes e depois da string afim de não recebermos erros do banco de dados, conforme mostra a **Listagem 10**.

O restante do método se comporta de maneira bastante semelhante ao FindAll, criando uma lista genérica de objetos que é populada a cada registro do DataSet que foi preenchido com os dados vindos da tabela CLIENTES.

Copyright - Proibido copiar ou distribuir. Todos os direitos reservados para DevMedia

Listagem 7. Classes TCliente e Tfornecedor

```
01 type
02  TCliente = class
03  private
04    FId: Integer;
05    FNome: string;
06    FCPF: string;
07  public
08    property Id: Integer read FId write FId;
09    property Nome: string read FNome write FNome;
10    property CPF: string read FCPF write FCPF;
11    constructor Create; overload;
12    constructor Create(AId: integer; ANome, ACPF: string); overload;
13  end;
14
15  TFornecedor = class
16  private
17    FId: Integer;
18    FNome: string;
19    FCNPJ: string;
20  public
21    property Id: Integer read FId write FId;
22    property Nome: string read FNome write FNome;
23    property CNPJ: string read FCNPJ write FCNPJ;
24    constructor Create(AId: integer; ANome: string; ACNPJ: string); overload;
25    constructor Create; overload;
26  end;
27
28 implementation
29
30 constructor TCliente.Create;
31 begin
32 end;
33
34 constructor TCliente.Create(AId: integer; ANome, ACPF: string);
35 begin
36  FId := AId;
37  FNome := ANome;
38  FCPF := ACPF;
39 end;
40
41 constructor TFornecedor.Create(AId: integer; ANome, ACNPJ: string);
42 begin
43  FId := AId;
44  FNome := ANome;
45  FCNPJ := ACNPJ;
46 end;
47
48 constructor TFornecedor.Create;
49 begin
50 end;
```

Listagem 8. Interface da Classe TClienteDAO

```
01 TClienteDAO = class(TDAO)
02  function Insert(ACliente: TCliente): Integer;
03  procedure Update(ACliente: TCliente);
04  procedure Delete(AId: Integer);
05  function FindById(AId: Integer): TCliente;
06  function FindByName(AName: string): TObjectList<TCliente>;
07 end;
```

Listagem 9. Métodos Insert, Update e Delete da classe TClienteDAO

```
01 function TClienteDAO.Insert(ACliente: TCliente): Integer;
02 var
03  SQL: string;
04  IdCliente: Integer;
05 begin
06  IdCliente := GetKeyValue('CLIENTES','ID_CLIENTE');
07  SQL := 'insert into CLIENTES values (:PAR1, :PAR2, :PAR3)';
08  Connection.ExecSQL(SQL, [ACliente.Id, ACliente.Nome, ACliente.CPF],
09    [ftInteger, ftString, ftString]);
10 end;
11
12 procedure TClienteDAO.Update(ACliente: TCliente);
13 var
14  SQL: string;
15 begin
16  SQL := 'update CLIENTES set NOME_CLIENTE = :PAR1, CPF = :PAR2 where
ID_CLIENTE = :PAR3';
17  Connection.ExecSQL(SQL, [ACliente.Nome, ACliente.CPF, ACliente.Id],
18    [ftString, ftString, ftInteger]);
19 end;
20
21 procedure TClienteDAO.Delete(AId: Integer);
22 var
23  SQL: string;
24 begin
25  SQL := 'delete from CLIENTES where ID_CLIENTE = ' + IntToStr(AId);
26  Connection.ExecSQL(SQL);
27 end;
```

A interface da classe TFornecedorDAO não será exibida no corpo deste artigo, mas pode ser consultada no arquivo fonte que acompanha a revista no site. Sua definição é muito semelhante a TClienteDAO, mudando-se apenas o tipo de objeto, de TCliente para TFornecedor.

Na **Listagem 11** temos a implementação de toda a classe TFornecedorDAO, que ficou com o código mais enxuto que o da TClienteDAO. Na linha 3, por exemplo, temos a execução do comando Delete de forma direta, sem uma variável auxiliar para armazenar o SQL. Já no método Insert não temos a criação da variável auxiliar para armazenar o identificador da tabela, pois estamos atribuindo diretamente na propriedade Id do objeto AFornecedor o retorno do método herdado GetKeyValue.

Na **Listagem 12** temos a classe TItemConta com sua interface e implementação. No SetNumParcela (linha 39) temos uma validação da informação que está sendo passada ao atributo FnumParcela, que somente aceita valores positivos.

Classe TContaDAO

A **Listagem 13** mostra as operações da classe DAO principal do sistema, pois engloba as duas principais classes de negócio, a TContaPagar e TContaReceber.

O método Delete é relativamente simples, pois somente chama uma exclusão da linha da tabela CONTAS, isso porque configuramos o banco de dados para fazer delete cascade entre as tabelas CONTAS e ITENS_CONTA, ou seja, quando excluirmos uma conta, automaticamente o banco de dados irá excluir todos os itens de conta vinculados a ela. Desta forma, reduzimos a codificação do nosso método, além de deixar a aplicação mais performática, pois o próprio banco se encarrega de fazer a exclusão em cascata.

DEVMEDIA Copyright - Proibido copiar ou distribuir. Todos os direitos reservados para DevMedia

Listagem 10. Métodos FindById e FindByName da classe TClienteDAO

```
01 function TClienteDAO.FindById(AId: Integer): TCliente;
02 var
03   Cliente: TCliente;
04   FDQuery: TFDQuery;
05 begin
06   FDQuery := TFDQuery.Create(nil);
07   try
08     FDQuery.Connection := Connection;
09     FDQuery.SQL.Text := 'select * from CLIENTES where ID_CLIENTE = ' + IntToStr(AId);
10     FDQuery.Open();
11     if FDQuery.RecordCount = 0 then
12       raise Exception.Create('Objeto não encontrado na base de dados!');
13     Cliente := TCliente.Create();
14     Cliente.Id := FDQuery.FieldByName('ID_CLIENTE').AsInteger;
15     Cliente.Nome := FDQuery.FieldByName('NOME_CLIENTE').AsString;
16     Cliente.CPF := FDQuery.FieldByName('CPF').AsString;
17   finally
18     FDQuery.Free;
19   end;
20   Result := Cliente;
21 end;
22
23 function TClienteDAO.FindByName(AName: string): TObjectList<TCliente>;
24 var
25   FDQuery: TFDQuery;
26   Clientes: TObjectList<TCliente>;
27 begin
28   FDQuery := TFDQuery.Create(nil);
29   try
30     FDQuery.Connection := Connection;
31     FDQuery.SQL.Text := 'select * from CLIENTES where NOME_CLIENTE like
       ' + QuotedStr('%'+AName+'%');
32     FDQuery.Open();
33     Clientes := TObjectList<TCliente>.Create();
34     while not FDQuery.Eof do
35     begin
36       Clientes.Add(TCliente.Create(FDQuery.FieldByName('ID_CLIENTE').AsInteger,
37         FDQuery.FieldByName('NOME_CLIENTE').AsString, FDQuery.
          FieldByName('CPF').AsString));
38     end;
39   finally
40     FDQuery.Free;
41   end;
42   result := Clientes;
43 end;
```

Listagem 11. Implementação da classe TFornecedorDAO

```
01 procedure TFornecedorDAO.Delete(AId: Integer);
02 begin
03   Connection.ExecSQL('delete from FORNECEDORES
       where ID_FORNECEDOR = ' + IntToStr(AId));
04 end;
05
06 function TFornecedorDAO.FindById(AId: Integer): TFornecedor;
07 var
08   Fornecedor: TFornecedor;
09   FDQuery: TFDQuery;
10 begin
11   FDQuery := TFDQuery.Create(nil);
12   try
13     FDQuery.Connection := Connection;
14     FDQuery.SQL.Text := 'select * from FORNECEDOR
       where ID_FORNECEDOR = ' + IntToStr(AId);
15     FDQuery.Open();
16     if FDQuery.RecordCount = 0 then
17       raise Exception.Create('Objeto não encontrado na base de dados!');
18     Fornecedor := TFornecedor.Create(FDQuery.FieldByName('ID_FORNECEDOR').
       AsInteger,
19       FDQuery.FieldByName('NOME_FORNECEDOR').AsString, FDQuery.
         FieldByName('CNPJ').AsString);
20   finally
21     FDQuery.Free;
22   end;
23   Result := Fornecedor;
24 end;
25
26 function TFornecedorDAO.FindByName(AName: string):
TObjectList<TFornecedor>;
27 var
28   FDQuery: TFDQuery;
29   Fornecedores: TObjectList<TFornecedor>;
30 begin
31   FDQuery := TFDQuery.Create(nil);
32   try
33     FDQuery.Connection := Connection;
34     FDQuery.SQL.Text := 'select * from FORNECEDORES
       where NOME_FORNECEDOR like ' +
35       QuotedStr('%' + AName + '%');
36     FDQuery.Open();
37     Fornecedores := TObjectList<TFornecedor>.Create();
38     while not FDQuery.Eof do
39     begin
40       Fornecedores.Add(TFornecedor.Create(FDQuery.FieldByName
         ('ID_FORNECEDOR').AsInteger,
41         FDQuery.FieldByName('NOME_FORNECEDOR').AsString,
           FDQuery.FieldByName('CNPJ').AsString));
42     end;
43   finally
44     FDQuery.Free;
45   end;
46   Result := Fornecedores;
47 end;
48
49 function TFornecedorDAO.Insert(AFornecedor: TFornecedor): Integer;
50 var SQL: string;
51 begin
52   AFornecedor.Id := GetKeyValue('FORNECEDORES','ID_FORNECEDOR');
53   SQL := 'insert into FORNECEDORES values (:PAR1, :PAR2, :PAR3)';
54   Connection.ExecSQL(SQL, [AFornecedor.Id, AFornecedor.Nome,
     AFornecedor.CNPJ],
55     [ftInteger, ftString, ftString]);
56 end;
57
58 procedure TFornecedorDAO.Update(AForncedor: TFornecedor);
59 var SQL: string;
60 begin
61   SQL := 'update FORNECEDORES set NOME_FORNECEDOR = :PAR1,
       CPF = :PAR2 where ID_FORNECEDOR = :PAR3';
62   Connection.ExecSQL(SQL, [AForncedor.Nome, AForncedor.CNPJ, AForncedor.Id],
63     [ftString, ftString, ftInteger]);
64 end;
```

Copyright - Proibido copiar ou distribuir. Todos os direitos reservados para DevMedia

Listagem 12. Classe TItemConta completa

```
01 TItemConta = class(TObject)
02 private
03   FId: Integer;
04   FNumParcela: Integer;
05   FNumDocumento: string;
06   FValor: Double;
07   FDataVencimento: TDate;
08 protected
09   procedure SetId(const Value: Integer);
10   procedure SetNumParcela(const Value: Integer);
11   procedure SetNumDocumento(const Value: string);
12   procedure SetValor(const Value: Double);
13   procedure SetDataVencimento(const Value: TDate);
14 public
15   property Id: Integer read FId write SetId;
16   property NumParcela: Integer read FNumParcela write SetNumParcela;
17   property NumDocumento: string read FNumDocumento write
       SetNumDocumento;
18   property Valor: Double read FValor write SetValor;
19   property DataVencimento: TDate read FDataVencimento write
       SetDataVencimento;
20 end;
21
22 { TItemConta }
23
24 procedure TItemConta.SetDataVencimento(const Value: TDate);
25 begin
26   FDataVencimento := Value;
27 end;
28
29 procedure TItemConta.SetId(const Value: Integer);
30 begin
31   FId := Value;
32 end;
33
34 procedure TItemConta.SetNumDocumento(const Value: string);
35 begin
36   FNumDocumento := Value;
37 end;
38
39 procedure TItemConta.SetNumParcela(const Value: Integer);
40 begin
41   if FNumParcela > 0 then
42     FNumParcela := Value;
43 end;
44
45 procedure TItemConta.SetValor(const Value: Double);
46 begin
47   FValor := Value;
48 end;
```

Listagem 13. Métodos Insert, Update e Delete

```
01 procedure TContaDAO.Delete(AId: Integer);
02 var
03   SQL: string;
04 begin
05   SQL := 'delete from CONTAS where ID_CONTA = ' + IntToStr(AId);
06   Connection.ExecSQL(SQL);
07 end;
08
09 function TContaDAO.Insert(AConta: TConta): Integer;
10 var
11   SQLConta, SQLItem: string;
12   IdContaGerado: Integer;
13   IdReferencia: Integer;
14   ItemConta: TItemConta;
15 begin
16   Connection.StartTransaction;
17   try
18     IdContaGerado := GetKeyValue('CONTAS', 'ID_CONTA');
19     SQLConta := 'insert into CONTAS values (:PAR1, :PAR2, :PAR3, :PAR4, :PAR5)';
20     AConta.ID := IdContaGerado;
21     if AConta is TContaPagar then
22       IdReferencia := (AConta as TContaPagar).Fornecedor.Id
23     else if AConta is TContaReceber then
24       IdReferencia := (AConta as TContaReceber).Cliente.Id;
25     Connection.ExecSQL(SQLConta, [AConta.ID, AConta.ValorTotal,
       AConta.DataLancamento,
26       AConta.Situacao, IdReferencia], [ftInteger, ftFloat, ftString, ftDate, ftInteger]);
27     for ItemConta in AConta.ItensConta do
28     begin
29       ItemConta.Id := GetKeyValue('ITENS_CONTA', 'ID_ITEM_CONTA');
30       SQLItem := 'insert into ITENS_CONTA values (:PAR1, :PAR2, :PAR3, :PAR4,
         :PAR5, :PAR6)';
31       Connection.ExecSQL(SQLItem, [ItemConta.Id, ItemConta.NumParcela,
32       ItemConta.NumDocumento, ItemConta.Valor, ItemConta.
         DataVencimento, AConta.ID],
33       [ftInteger, ftInteger, ftString, ftFloat, ftDate, ftInteger]);
34     end;
35     Connection.Commit;
36     except
37       Connection.Rollback;
38     end;
39   end;
40
41   procedure TContaDAO.Update(AConta: TConta);
42   var
43     SQLConta, SQLItem: string;
44     ItemConta: TItemConta;
45     IdReferencia: integer;
46   begin
47     Connection.StartTransaction;
48     try
49       SQLConta := 'update CONTAS set VALOR_TOTAL = :PAR1,
         SITUACAO_CONTA = :PAR2, ' +
50 '       DATA_LANCAMENTO = :PAR3, ID_REFERENCIA = :PAR4
          where ID_CONTA = :PAR5';
51     if AConta is TContaPagar then
52       IdReferencia := (AConta as TContaPagar).Fornecedor.Id
53     else if AConta is TContaReceber then
54       IdReferencia := (AConta as TContaReceber).Cliente.Id;
55     Connection.ExecSQL(SQLConta, [AConta.ValorTotal, AConta.Situacao,
56     AConta.DataLancamento, IdReferencia, AConta.ID]);
57     for ItemConta in AConta.ItensConta do
58     begin
59       SQLItem := 'update ITENS_CONTA set NUM_PARCELA = :PAR1,
         NUM_DOCUMENTO = :PAR2, ' +
60 '     VALOR_PARCELA = :PAR3, DATA_VENCIMENTO = :PAR4, ID_CONTA = :PAR5 ' +
61       'where ID_ITEM_CONTA = :PAR6';
62       Connection.ExecSQL(SQLItem, [ItemConta.NumParcela,
         ItemConta.NumDocumento,
63       ItemConta.Valor, ItemConta.DataVencimento, AConta.ID, ItemConta.Id],
64       [ftInteger, ftString, ftFloat, ftDate, ftInteger, ftInteger]);
65     end;
66     Connection.Commit;
67     except
68     Connection.Rollback;
69     end;
70   end;
```


DEVMEDIA Copyright - Proibido copiar ou distribuir. Todos os direitos reservados para DevMedia

No método Insert temos a utilização de um importante recurso chamado transação (**BOX 3**), isso porque precisamos garantir que nossos dados fiquem de forma íntegra na base de dados. Abrimos uma transação chamando o método StartTransaction da classe TFDConnection e, em seguida contornamos o restante dos procedimentos de atualização com um bloco try/except, de maneira que, caso ocorra algum erro durante este processo é chamado o método Rollback, caso contrário, é chamado o método Commit, que confirma todas as alterações feitas no banco.

Pelo fato de termos uma única tabela para contas a pagar e a receber, se faz necessário o uso de uma variável chamada IdReferencia, isto porque a tabela CONTAS deverá armazenar uma referência para CLIENTES quando for uma conta a receber e uma referência para FORNECEDORES quando for uma conta a receber.

Desta vez o método Insert fará operações de inserção em duas tabelas, por isso temos duas vezes a chamada do GetKeyValue: um para a tabela CONTAS e vários quantos forem os itens para a tabela ITENS_CONTA, dentro de um foreach que irá percorrer todos os itens de conta.

O método Update funciona de maneira semelhante ao método Insert e também precisamos executá-lo dentro de uma transação, afim de garantir a total integridade dos dados.

Neste artigo pudemos observar o quanto é árduo o trabalho de codificarmos as classes DAO, de maneira a trabalhar totalmente orientado a objetos. Lembre-se que ainda temos as classes de Modelo na interface com o usuário que correspondem a camada de visão ou View.

Até a próxima.

BOX 3. Transações

Transações são um conjunto de operações atômicas que devem ser executadas pelo banco de dados e existe para garantirmos a integridade dos dados após uma série atualizações, garantindo que todas ocorreram com sucesso. No caso de uma falha, todo o processo é cancelado, trazendo grande segurança ao sistema. Controle de transações é algo obrigatório em sistemas de informações.

-ReadCommited: nível de isolamento padrão, onde as transações têm acesso a apenas dados efetivados, que foram feitos commit, nunca acessa dados que ainda não foram confirmados em outras transações.

-RepeatableRead: este nível enxerga os dados apenas no início da transação e não tem acesso a dados não confirmados por outras transações.

-DirtyRead: nível de isolamento mais baixo, permitindo que a transação corrente leia dados ainda não confirmados por outras transações, mesmo sem o commit.

-Serializable: maior nível de isolamento, onde todas as operações devem ser feitas de forma totalmente isolada, uma após a outra.

Autor

FILIPE DALEPIANE

filipe.dalepiane@gmail.com

Bacharel em Ciência da Computação, certificado Delphi Developer, colunista da revista Clube Delphi e .NET Magazine. Desenvolve em Delphi para Desktop e Mobile e C# para Web. Trabalha atualmente como Analista de Sistemas na AVMB Consultoria e Assessoria em Informática em Santa Maria-RS (www.avmb.com.br).

Você gostou deste artigo?

Dê seu voto em ***www.devmedia.com.br/clubedelphi/feedback***

Ajude-nos a manter a qualidade da revista!

Conhecimento faz diferença!

Faça já sua assinatura digital ! | www.devmedia.com.br/es

Faça um *upgrade* em sua carreira

Em um mercado cada vez mais focado em qualidade, ter conhecimentos aprofundados sobre requisitos, metodologia, análises, testes, entre outros, pode ser a diferença entre conquistar ou não uma boa posição profissional. Sabendo disso a DevMedia lançou uma publicação totalmente especializada em Engenharia de Software. Todos os meses você pode encontrar artigos sobre Metodologias Ágeis; Metodologias tradicionais (document driven); ALM (application lifecycle); SOA (aplicações orientadas a serviços); Análise de sistemas; Modelagem; Métricas; Orientação à Objetos; UML; testes e muito mais. **Assine Já!**

DEVMEDIA

Explorando a API do Windows no Delphi – Parte 1

Conheça e utilize recursos do Windows

Este artigo faz parte de um curso

As APIs do Windows são expostas através de DLLs que podem ser utilizadas no Delphi e quando as utilizamos estamos lidamos diretamente com o sistema operacional. Dentre as categorias existentes nas APIs, pode-se dizer que as principais são:

- Windows;
- Arquivos;
- Informações sobre o sistema;
- Cursores;
- Mensagens;
- Mouse;
- Teclado;
- Impressoras;
- Ícones;
- Arquivos INI;
- Registro;
- Dispositivos;
- Acessibilidade.

A Embarcadero disponibiliza no Delphi o acesso a essas APIs através da unit Windows, que realiza uma ponte entre o código Delphi e as várias DLLs disponibilizadas pelo sistema. As principais DLLs são:

- User32.dll;
- kernel32.dll;
- Comdlg32.dll;
- gdi32.dll;
- shell32.dll;
- Advapi32.dll;
- winmm.dll.

Fique por Dentro

Recursos como o de criar pastas, achar determinado arquivo, apagar temporários após sua utilização, entre outros, estão disponíveis pela API do Windows, que é um conjunto de DLLs que fazem parte do sistema, expondo as funções do mesmo. Nesse artigo vamos explorar as APIs da categoria Arquivos, Cursores, Registros e Informações sobre o sistema e Windows.

Uma DLL (Dynamic-link library ou biblioteca de vínculo dinâmico), é um arquivo com extensão que consiste numa coleção de funções e procedures que podem ser chamadas por outras aplicações e outras DLLs, que por sua vez, é ligada em tempo de execução ao programa que as usa.

Informações sobre o Sistema

É possível obter informações sobre o Sistema através de algumas funções expostas:

- GetComputerName: está declarada em kernel32.dll e irá ler o nome do computador, que será devolvido em uma variável do tipo string. Esta deve ser passada como parâmetro na função. Sua declaração é feita da seguinte forma:

```
GetComputerNameA (ByVal lpBuffer As String, nSize As Long) As Long
```

O parâmetro lpBuffer é uma sequência de caracteres que deve ser grande o suficiente para manter o nome do computador. Já nSize é o comprimento em caracteres de lpBuffer, geralmente usado com o valor 255.

- **GetUserName**: está declarada em **advapi32.dll** e recupera o nome do usuário que está logado no Windows. Este também é retornado em uma string que devemos passar como parâmetro. Sua declaração é a seguinte:

 Copyright - Proibido copiar ou distribuir. Todos os direitos reservados para DevMedia

```
GetUserNameA (ByVal lpBuffer As String, nSize As Long) As Long
```

O lpBuffer é uma sequência de caracteres que deve ser grande o suficiente para manter o nome do usuário. O nSize é o comprimento em caracteres de lpBuffer, geralmente com o valor 144.

• **GetSystemDirectory:** retorna o caminho do diretório de sistema do Windows. É importante observar é que nunca devemos assumir que o diretório é "C:\Windows\System", porque o diretório não necessariamente precisa ser chamado Windows. Sua declaração é parecida com as outras duas que vimos anteriormente e até mesmo os mesmos parâmetros são parecidos. Ela está declarada em **kernel32.dll** e sua declaração é:

```
GetSystemDirectoryA (ByVal lpBuffer As String, ByVal nSize As Long) As Long
```

• **GetWindowsDirectory:** está declarada em **kernel32.dll** e retorna o caminho do diretório do Windows. É onde o próprio Windows está instalado, contudo, isso não significa que seja sempre "C:\Windows". Sua declaração é:

```
GetWindowsDirectoryA (ByVal lpBuffer As String, ByVal nSize As Long) As Long
```

• **GetTempPath:** retorna o diretório Temp do Windows, onde ficam os arquivos temporários. A função está declarada em **kernel32.dll** e, ao contrário das quatro funções vistas anteriormente, aqui os parâmetros se invertem. Primeiro é passado o tamanho a ser usado para receber a string, e depois o parâmetro da mesma, como na declaração a seguir:

```
GetTempPathA (ByVal nBufferLength As Long, ByVal lpBuffer As String) As Long
```

• **GetVersionEx**: declarada em **kernel32.dll**, esta função retorna as informações sobre a versão do Windows em execução. Essas informações incluem o número da versão, o build e a versão do sistema instalado. Essas informações são transferidas para uma variável do tipo OSVersionInfo, que é do tipo record, conforme a sua declaração:

```
GetVersionExA (lpVersionInformation As OSVERSIONINFO) As Long
```

Aplicação sobre Informações do Sistema

Para mostrar como utilizar essas APIs vamos desenvolver uma aplicação. Nela teremos apenas uma tela com seis TEdits, seis TLabels e um TButton, como vemos na **Figura 1.**

A propriedade Name do Form1 é modificada para Frm_Principal, e os seis TEdits para Edt_CompNome, Edt_UsurNome, Edt_PastaSys, Edt_WinDiretorio, Edt_PastaTemp e Edt_WinVersao. O botão recebe o nome de Btn_Informacoes e os seis TLabels têm sua propriedade name modificada para Lbl_CompNome, Lbl_UsurNome, Lbl_PastaSys, Lbl_WinDiretorio, Lbl_PastaTemp, e Lbl_WinVersao. Ao salvar a aplicação ajustamos a unit para o nome de Unt_Principal e o projeto para InfoSys. Já a propriedade Caption dos Tlabels deve ficar como visto na **Figura 1**.

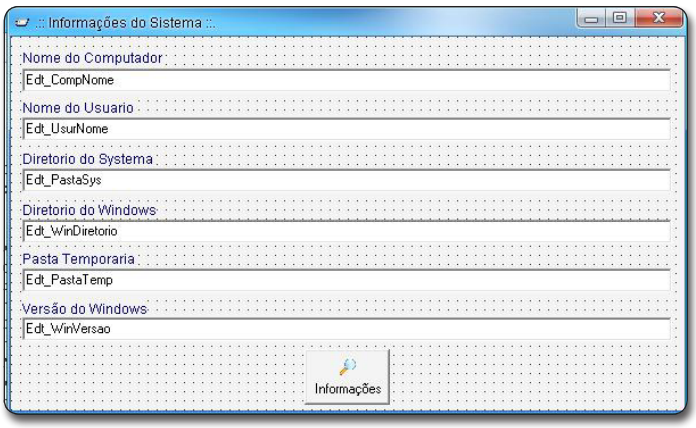


Figura 1. Tela do aplicativo

Procedimentos e Funções da Seção Private

Na seção private são declaradas seis funções, como mostra a **Listagem 1**. Elas são responsáveis por acessar as APIs. Uma vez declaradas pressionamos a combinação Shift + Ctrl + C e com isso o Delphi inicia a implementação dessas funções, que podemos ver no código da **Listagem 2**.

Listagem 1. Seção Private do Frm_Principal

```
private
  function fGetComputerName: String;
  function fGetUserName: String;
  function fGetSystemDirectory: String;
  function fWindowsDirectory: String;
  function fGetTempPath: String;
  function fGetVersionEx: string;
```

As funções a seguir foram criadas para obter informações e repassá-las aos controles TEdit:

• StrPas – É a função declarada na Unit SysUtils que converte uma cadeia de strings, terminado em nulo, para uma cadeia de string longa (AnsiString).

• SizeOf – É a função declarada na Unit System, que retorna o tamanho em bytes de uma variável ou tipo.

• TOSVersionInfo – É um record declarado em SysUtils, que contém informações do sistema operacional, plataforma (Windows, Mac Os X), versão, tipo de arquitetura (Intel x86 ou Intel x64) e Service Pack. Esse record contém dois tipos públicos:

- TArchitecture (arIntelX86, arIntelX64, arARM32);
- TPlatform (pfWindows, pfMacOS, pfiOS, pfAndroid, pfWinRT, pfLinux).

Arquivos

O Windows oferece uma grande variedade de funções para tratamento de arquivos como vemos a seguir:

Copyright - Proibido copiar ou distribuir. Todos os direitos reservados para DevMedia

• **CopyFile:** está declarada em kernel32.dll e copia um arquivo de um local para outro, assim como a cópia de um arquivo no Windows Explorer. Em sua declaração temos três parâmetros:

- LpExistingFileName - O arquivo de origem, ou seja, o arquivo a ser copiado;
- LpNewFileName - O arquivo de destino, ou seja, o novo arquivo para criar;
- BFailIfExists - Se 0, a função irá substituir LpNewFileName caso ele já existe, caso contrário, a função irá falhar.

Listagem 2. Implementação

```
procedure TFrm_Principal.Btn_InformacoesClick(Sender: TObject);
begin
  Edt_CompNome.Text    := fGetComputerName;
  Edt_UsurNome.Text    := fGetUserName;
  Edt_PastaSys.Text    := fGetSystemDirectory;
  Edt_WinDiretorio.Text := fWindowsDirectory;
  Edt_PastaTemp.Text   := fGetTempPath;
  Edt_WinVersao.Text   := fGetVersionEx;
end;

function TFrm_Principal.fGetComputerName: String;
var
  Buffer: Array[0..255] of Char;
  I: DWord;
begin
  I := SizeOf(Buffer);
  GetComputerName(Buffer, I);
  Result := StrPas(Buffer);
end;

function TFrm_Principal.fGetSystemDirectory: String;
var
  Buffer: Array[0..255] of Char;
begin
  GetSystemDirectory(Buffer, 255);
  Result := StrPas(Buffer);
end;

function TFrm_Principal.fGetTempPath: String;
var
  Buffer: Array[0..255] of Char;
begin
  GetTempPath(255, Buffer);
  Result := StrPas(Buffer);
end;

function TFrm_Principal.fGetUserName: String;
var
  Buffer: Array[0..255] of Char;
  I: DWord;
begin
  I := SizeOf(Buffer);
  GetUserName(Buffer, I);
  Result := StrPas(Buffer);
end;

function TFrm_Principal.fWindowsDirectory: String;
var
  Buffer: Array[0..255] of Char;
begin
  GetWindowsDirectory(Buffer, 255);
  Result := StrPas(Buffer);
end;

function TFrm_Principal.fGetVersionEx: string;
var
  VersionInfo: TOSVersionInfo;
begin
  VersionInfo.dwOSVersionInfoSize := SizeOf(VersionInfo);

  GetVersionEx(VersionInfo);

  with VersionInfo do
begin
  case dwPlatformid of
    0: begin
       Result := 'Windows 3.11';
     end;

    1: begin
       case dwMinorVersion of
         0: Result := 'Windows 95';

         10: begin
             if (szCSDVersion[ 1 ] = 'A' ) then
               Result :='Windows 98 SE'
             else
               Result := 'Windows 98';
           end;

         90: Result := 'Windows Millenium';
         else
          Result := 'Não achei a Versão';
       end;
     end;

    2: begin
       case dwMajorVersion of
         3: Result := 'Windows NT ' + IntToStr(dwMajorVersion) + '.' +
                 IntToStr(dwMinorVersion);

         4: Result := 'Windows NT ' + IntToStr(dwMajorVersion) + '.' +
                 IntToStr(dwMinorVersion);

         5: begin
            case dwMinorVersion of
              0: Result := 'Windows 2000';
              1: Result := 'Windows XP';
            end;
          end;

         6: Result := 'Windows 7 ' + IntToStr(dwMajorVersion) + '.' +
                 IntToStr(dwMinorVersion);

         7: Result := 'Windows 8 ' + IntToStr(dwMajorVersion) + '.' +
                 IntToStr(dwMinorVersion);

         8: Result := 'Windows Vista ' + IntToStr(dwMajorVersion) + '.' +
                 IntToStr(dwMinorVersion);
         else
           Result := 'Não achei a Versão';
       end;

       if szCSDVersion <> '' then
         Result := Result + ' ' + szCSDVersion;
     end;
    else
      Result := 'Não achei a Platforma';
  end;

  Result := Result + ', Build: ' + IntToStr(Loword(dwBuildNumber)) ;
 end;
end;

end.
```


DEVMEDIA Copyright - Proibido copiar ou distribuir. Todos os direitos reservados para DevMedia

• **MoveFile:** move ou renomeia um arquivo ou pasta. Se um diretório é movido/renomeado, todos os subdiretórios e arquivos contidos nele serão afetados. A função retorna 1 se for bem-sucedida ou zero se ocorrer um erro. Espera-se dois parâmetros:

- LpExistingFileName - O arquivo de origem ou diretório, ou seja, o arquivo ou diretório para renomear (mover);
- LpNewFileName - O arquivo de destino ou diretório, ou seja, o novo nome do arquivo ou diretório, que se dá ao arquivo de origem para que seja movido.

• **CreateFile:** cria ou abre um arquivo em disco para acesso posterior. É necessário ter os direitos de acesso permitidos para o arquivo a ser utilizado. Essa função tem inúmeros parâmetros para especificar os níveis e tipos de acesso, e irá retornar o identificador para o arquivo criado/aberto se for bem-sucedido, ou -1 se um algum erro ocorreu. Seus parâmetros são:

- LpFileName - O nome do arquivo a ser criado ou aberto;
- DwDesiredAccess - Zero ou um dos seguintes parâmetros, especificando as quantidades de acesso, de leitura e gravação para o arquivo:
  - GENERIC_READ - Permitir que o programa leia os dados do arquivo;
  - GENERIC_WRITE - Permitir que o programa grave dados no arquivo.
- DwShareMode - Zero ou um dos seguintes parâmetros, especificando as quantidades de acesso, de leitura e gravação concedidas a outros programas enquanto o programa ainda está com ele aberto:
  - FILE_SHARE_READ - Permitir que outros programas possam ler os dados do arquivo;
  - FILE_SHARE_WRITE - Permitir que outros programas possam gravar dados no arquivo.
- LpSecurityAppributes - Os atributos de segurança dados ao arquivo criado ou aberto;
- DwCreationDisposition - Exatamente um dos seguintes parâmetros, especificando como e quando criar ou abrir o arquivo, dependendo se ele já existe ou não:
  - CREATE_ALWAYS - Cria um novo arquivo, substituindo o mesmo caso esse já exista;
  - CREATE_NEW - Cria um novo arquivo, mas falha se ele já existe;
  - OPEN_ALWAYS - Abre um arquivo existente e, se o arquivo não existir, ele será criado;
  - OPEN_EXISTING - Abre um arquivo existente, mas falha se o arquivo não existe.
  - TRUNCATE_EXISTING - Abre um arquivo existente e apaga o seu conteúdo. A função falhará se o arquivo não existe.
- DwFlagsAndAttributes - Uma combinação dos seguintes parâmetros, especificando os atributos do arquivo para um recém-criado, para cria-lo ou abri-lo. Deve ser incluso um handle para o arquivo, para especificar os seus atributos:
  - FILE_ATTRIBUTE_ARCHIVE - Um arquivo normal;
  - FILE_ATTRIBUTE_HIDDEN - Um arquivo oculto, que normalmente não é visível para o usuário, dependendo da configuração;
  - FILE_ATTRIBUTE_NORMAL - Um arquivo sem atributos (esse não pode ser usado com atributos);
  - FILE_ATTRIBUTE_READONLY - Um arquivo de somente leitura;
  - FILE_ATTRIBUTE_SYSTEM - Um arquivo de sistema, utilizado exclusivamente pelo sistema operacional;
  - FILE_FLAG_DELETE_ON_CLOSE - Exclui o arquivo, uma vez que o mesmo está fechado;
  - FILE_FLAG_OVERLAPPED - Permiti que o arquivo seja lido e gravado ao mesmo tempo. Se for utilizado, as funções que leem e escrevem no arquivo devem especificar essa estrutura (OVERLAPPED) para identificar o ponteiro do arquivo;
  - FILE_FLAG_POSIX_SEMANTICS - Permiti que o nome do arquivo seja maiúsculo ou minúsculo;
  - FILE_FLAG_RANDOM_ACCESS - Otimiza o cache de arquivos para acesso aleatório (poder pular por várias partes do arquivo);
  - FILE_FLAG_SEQUENTIAL_SCAN - Otimiza o cache de arquivos para acesso sequencial crescente);
  - FILE_FLAG_WRITE_THROUGH – Lê e escreve diretamente no arquivo, ignorando qualquer cache de disco;
- HTemplateFile – Identifica 1 arquivo aberto e copia os atributos, ou zero para não copiar os mesmos.

• **DeleteFile:** exclui um arquivo completamente, sem enviá-lo para a lixeira. Ele também não solicita a confirmação da exclusão, então deve ser utilizado com toda atenção. A função retorna 1 se for bem-sucedida, ou zero caso tenha ocorrido algum erro como, por exemplo, quando o arquivo a ser excluído não existe. A função espera por um único parâmetro LpFileName, que representa o nome do arquivo a ser excluído;

• **FindClose**: termina a pesquisa de um arquivo iniciado por FindFirstFile. Esta função fecha o identificador da pesquisa de arquivos;

• **FindFirstFile:** começa uma pesquisa de arquivo e fornece informações sobre o primeiro arquivo correspondente. As pesquisas de arquivos têm base em apenas um nome de arquivo com sua extensão ou não. A pesquisa só olha em um único diretório, mas identifica quaisquer nomes no mesmo que corresponde à sequência da pesquisa A função retorna um identificador de pesquisa que pode ser usado para procurar por arquivos correspondentes adicionais, usando FindNextFile. Pode ser retornado -1 também caso não haja arquivos coincidentes com a pesquisa. Seus parâmetros são:

- LpFileName – É a sequência de pesquisa de arquivos para procurar, incluindo o caminho completo. Pode conter os curingas como * ou ?;
- LpFindFileData - Recebe informações de identificação sobre o primeiro arquivo encontrado.

• **FindNextFile:** continua uma pesquisa de arquivo que começou com FindFirstFile. Encontra e fornece informações de identificação

Copyright - Proibido copiar ou distribuir. Todos os direitos reservados para DevMedia

sobre o próximo arquivo que corresponde à sequência da pesquisa. A função retorna 1 se foi encontrado um outro arquivo, ou zero se não existem mais arquivos correspondentes (ou se ocorreu um erro). Seus parâmetros são:

- HFindFile - identificador do arquivo para a pesquisa começado com FindFirstFile;
- LpFindFileData - recebe informações de identificação sobre o próximo arquivo correspondente que foi encontrado.

• **CreateDirectory:** cria um novo diretório em disco e define os atributos de segurança do mesmo. A função retorna 1 se for bem-sucedida, ou zero se ocorrer algum erro. Seus parâmetros são:

- LpPathName - nome do novo diretório a ser criado;
- LpSecurityAttributes - atributos de segurança para dar ao novo diretório.

• **FileExists:** retorna um valor boolean se o arquivo especificado como parâmetro existe ou não.

• **DirectoryExists:** retorna um valor boolean se o diretório especificado existir ou não. Seu único parâmetro é o Directory, que representa o nome do diretório para verificar sua existência. A função poderá falhar se o usuário não tiver permissão para o caminho informado do diretório.

• **CloseHandle:** fecha um identificador e o objeto associado a ele. Depois de ter sido fechado, o identificador não será mais válido. A função retorna 1 se for bem-sucedida, ou zero se ocorreu algum erro. Seu parâmetro HObject representa o identificador do objeto a fechar.

Aplicação sobre Arquivos

Ao criar um novo aplicativo do tipo Win32, adicionamos ao formulário principal seis componentes TLabel, seis TButton e um TListBox, como mostra a **Figura 2.**

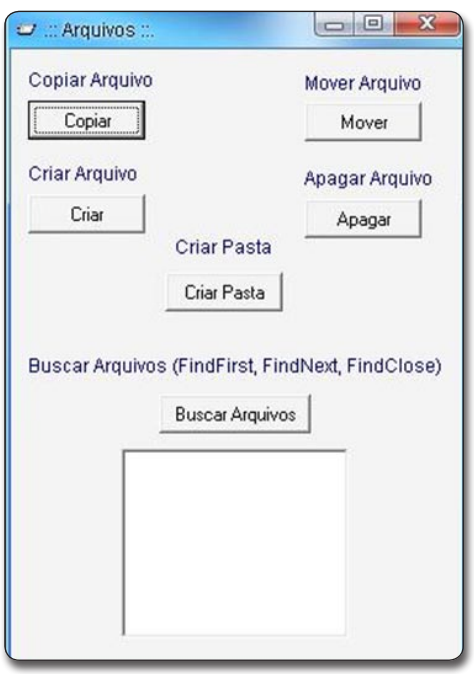


Figura 2. Tela do Aplicativo

A propriedade Name do Form1 é modificada para Frm_Principal e os seis TButons para Btn_Copiar, Btn_Mover, Btn_Criar, Btn_Apagar, Btn_CriaDir e Btn_BuscArqui. O TListBox é chamado de Lst_BuscArqui, e os seis TLabel devem ter a propriedade Name modificada para Lbl_Copiar, Lbl_Mover, Lbl_Apagar, Lbl_Criar, Lbl_CriaDir e Lst_BuscArqui. Salvamos a unit com o nome de Unt_Principal e o projeto como Arquivos. Mude também a propriedade Caption dos TLabels e TButtons conforme a **Figura 2**.

Incluímos na seção uses a unit ShellApi, pois precisaremos dela para criar, copiar, mover, apagar os arquivos. Usaremos arquivos de textos normais (*.txt).

A **Listagem 3** mostra a implementação do botão Buscar Arquivos.

TSearchRec é um record que está declarado em SysUtils e define uma estrutura de dados, que veremos na **Listagem 4**. Ela será utilizada para armazenar informações de pesquisa de arquivos pelas rotinas FindFirst e FindNext.

Listagem 3. Usando as funções e implementando o código dos botões

```
procedure TFrm_Principal.Btn_BuscArquiClick(Sender: TObject);
var
  SR: TSearchRec;
  Pasta: String;
begin
  Pasta := 'C:\Apagar Arquivos Txt';

  if not(DirectoryExists(Pasta)) then
  begin
    Application.MessageBox('Não Existe a Pasta "C:\Apagar Arquivos Txt"',
            ' Atenção',MB_ICONINFORMATION + MB_OK);
    Exit;
  end;

  If FindFirst(Pasta +'\*.txt', faAnyFile, SR) =0 then
  begin
    Repeat
      if (SR.Attr and faDirectory) <> faDirectory then
        Lst_BuscArqui.Items.Add(Sr.Name);

    Until FindNext(SR) <> 0;

    FindClose(SR);
  end;
end;
```

Listagem 4. Estrutura TSearchRec

```
TSearchRec = record
  Time: Integer;
  Size: Integer;
  Attr: Integer;
  Name: TFileName;
  ExcludeAttr: Integer;
  FindHandle: THandle;
  FindData: TWin32FindData;
end;
```

Na estrutura record temos:

• Time: Data do arquivo modificado e o tempo;
• Size: Tamanho do arquivo em bytes;
• Attr: Atributos do arquivo:
  - faAnyFile: Qualquer arquivo;
  - faReadOnly: Arquivos somente leitura;

DEVMEDIA Copyright - Proibido copiar ou distribuir. Todos os direitos reservados para DevMedia

- faHidden: Arquivos ocultos;
- faSysFile: Os arquivos de sistema;
- faVolumeID: Volume: arquivos ID;
- faDirectory: Arquivos Diretório;
- faArchive: Arquivos;

• Name: Nome do arquivo.

Windows

A categoria Windows permite interação com suas janelas, habilitar e desabilitar diversas configurações e muito mais. Essa API conta com 31 funções, mas para esse artigo apresentaremos apenas as principais que serão utilizadas na aplicação de exemplo para entendermos mais sobre.

ShowWindow

A função pode minimizar, maximizar ou restaurar uma determinada janela. Retorna zero se a janela estiver invisível antes da chamada, ou um valor diferente se estiver visível. Recebe os seguintes parâmetros de entrada:

• Hwnd - O identificador da janela, para alterar o status de como é mostrado;

• NcmdShow - Exatamente um dos seguintes parâmetros, especificando como mostrar a janela:

- SW_HIDE - Esconde a janela;
- SW_MAXIMIZE - Maximiza a janela;
- SW_MINIMIZE - Minimiza a janela;
- SW_RESTORE - Restaura a janela (não maximizada e nem minimizada);
- SW_SHOW - Mostra a janela;
- SW_SHOWMAXIMIZED - Mostra a janela maximizada;
- SW_SHOWMINIMIZED - Mostra a janela minimizada;
- SW_SHOWMINNOACTIVE - Mostra a janela minimizada, mas não a ativa;
- SW_SHOWNA - Mostra a janela em seu estado atual, mas não a ativa;
- SW_SHOWNOACTIVATE -Mostra a janela em seu tamanho e a posição mais recente, mas não ativa;
- SW_SHOWNORMAL - Mostra a janela e a ativa (geralmente o normal).

FindWindow

Esta função procura por todas as janelas abertas que correspondam ao nome da classe da janela informado e/ou nome da janela. Essa busca não é sensível a maiúsculas e seus parâmetros são relacionados a seguir:

• LpClassName - O nome da classe da janela para se encontrar. Passe zero para permitir que a janela seja de qualquer classe;

• LpWindowName - O texto da barra de título da janela para se encontrar. Passe zero para permitir que a janela tenha qualquer nome.

Se ocorrer algum erro, ou uma janela correspondente não puder ser encontrada, a função retorna zero. Caso contrário, a função retornará um identificador para a janela encontrada.

GetForegroundWindow

Esta função acha a janela que está atualmente em primeiro plano. A janela em primeiro plano é a janela geralmente na qual o usuário está atualmente trabalhando, ou seja, a janela com o foco. A função retorna zero se um erro ocorrer, ou o identificador da janela se bem-sucedido.

GetWindowText

Retorna o texto da barra de título de uma janela. Esta função funciona com qualquer janela, não apenas aquelas em sua aplicação. O texto é devolvido em uma variável do tipo String, passada como parâmetro. A função também retorna o comprimento do texto, se bem-sucedida, ou zero se ocorreu algum erro. Seus parâmetros são:

• Hwnd - A janela para ler o título;

• LpString - Variável que recebe o texto da barra de título da janela;

• CCH - O comprimento em caracteres de LpString, ou seja, a quantidade de caracteres do título da janela.

GetWindowTextLength

Retorna o comprimento em caracteres do texto da barra de título de uma janela, ou retorna zero se ocorrer erro. Seu único parâmetro é Hwnd, que deve receber o identificador da janela a ser lida.

EnableWindow

Essa função ativa ou desativa uma janela. Se estiver desativada, ela não pode receber o foco e irá ignorar qualquer tentativa de entrada. Alguns tipos de controles, como botões, aparecerão desativados, embora qualquer janela possa ser ativada ou desativada. A função retorna zero se a janela está ativada, ou um valor diferente de zero se a janela está desativada. Recebe dois parâmetros:

• Hwnd -Um identificador para a janela a ser ativada ou desativada;

• FEnable - Se zero, a janela será desativada, caso contrário, a janela será ativada.

SetWindowPos

A função move uma janela para um novo local na tela. Suas coordenadas físicas, dimensões e posição, bem como o Z-order, que determina se a janela está em cima das outras, podem ser definidos. A função retorna zero caso ocorra um erro. A relação de seus parâmetros pode ser vista a seguir.

• Hwnd – Move a janela;

• HwndInsertAfter – É o identificador da janela para posicionar esta janela para trás. Um dos seguintes parâmetros pode ser passado, indicando onde Z-ordem deve colocar a janela:

- HWND_BOTTOM - Coloca a janela na parte inferior;
- HWND_NOTOPMOST - Coloca a janela abaixo de todas as janelas de nível superior, e acima de todas as janelas não-superiores;
- HWND_TOP - Coloca a janela na parte superior;

Copyright - Proibido copiar ou distribuir. Todos os direitos reservados para DevMedia

- HWND_TOPMOST - Coloca a janela no topo, ou seja, acima de todas as outras janelas, de forma permanente;

• X - A coordenada x do canto superior esquerdo da janela;
• Y - A coordenada y do canto superior esquerdo da janela;
• CX - A coordenada x do canto inferior direito da janela;
• CY - A coordenada y do canto inferior direito da janela
• Wflags - Zero ou um dos seguintes parâmetros, afirmando como mover a janela:

- SWP_DRAWFRAME - O mesmo que SWP_FRAMECHANGED;
- SWP_FRAMECHANGED - Redesenha totalmente a janela em sua nova posição;
- SWP_HIDEWINDOW - Ocultar a janela da tela;
- SWP_NOACTIVATE – Não ativa a janela após movê-la, a menos que a mesma já esteja ativa;
- SWP_NOCOPYBITS - Não redesenha nada na janela depois que for movida;
- SWP_NOMOVE - Não move a janela;
- SWP_NOSIZE - Não redimensiona a janela;
- SWP_NOREDRAW - Não remove a imagem da janela em sua antiga posição, efetivamente deixando uma imagem fantasma da tela;
- SWP_NOZORDER - Não muda a posição da janela no Z-ordem;
- SWP_SHOWWINDOW - Mostra a janela, caso esteja oculta.

IsIconic

Esta função verifica se uma determinada janela está minimizada ou não. A função retorna zero se a janela estiver minimizada ou um valor diferente se a janela não estiver minimizada. Seu único parâmetro de entrada é Hwnd, onde deve ser passado o identificar da janela a ser verificada.

IsZoomed

Verifica se uma determinada janela está maximizada ou não. A função retorna zero se a janela estiver maximizada ou retornará um valor diferente se a janela não estiver maximizada. Também tem como único parâmetro de entrada o Hwnd.

Aplicação usando a categoria Windows

Nesse exemplo criamos uma nova aplicação do tipo Win32 e em seu formulário principal adicionamos cinco controles TLabel e cinco TButton, como mostra a **Figura 3.** Para demonstrar as funções, vamos interagir com a janela dos aplicativos calculadora e bloco de notas do próprio Windows.

Modificamos a propriedade Name do Form1 para Frm_FWindows e os cinco controles TButton para Btn_ShowWin, Btn_Janela, Btn_EstJanela, Btn_MovJanela, e Btn_HabDesab. Cada botão possui um componente TLabel associado, que deve mudar a propriedade name desses controles para bl_ShowWin, Lbl_Janela, Lbl_EstJanela, Lbl_MovJanela e Lbl_HabDesab. Podemos então salvar a aplicação e, a unit com o nome de Unt_Principal e o projeto como PWindows. A **Listagem 5** mostra como podemos

Figura 3. Tela do Aplicativo

utilizar cada uma das APIs da categoria Windows detalhadas no artigo.

Para testar o código é preciso antes abrir o bloco de notas e a calculadora do Windows. O botão Btn_ShowWinClick faz uso da API FindWindow. Observe que passamos o nome da janela para a função. Se seu Windows estiver com o idioma inglês definido, por exemplo, e seus aplicativos também estiverem nessa língua, você deve passar o que aparece na barra de título. Ao ser encontrada, é retornado então seu identificador (handle), que é então passado para a função ShowWindow, que irá enviar o comando de exibição.

Na procedure Btn_JanelaClick utilizamos várias funções para obter informações da janela ativa. A primeira delas é a GetForegroundWindow, que retorna o identificador da janela em foco. Esse é o ponto de partida, porque uma vez com este em mãos, podemos acessar a janela. Na sequência usamos GetWindwTextLength para contar a quantidade de caracteres do título da janela ativa e depois, através de GetWindowText, obtemos qual é o título. É importante notar que o retorno de GetWindowText é armazenado e verificado. Se esse for maior que zero é porque o título foi recuperado com sucesso.

Na próxima procedure utilizamos IsIconic e IsZoomed para identificar se a janela está respectivamente minimizada ou maximizada. Já na procedure Btn_MovJanelaClick conseguimos mover a janela da calculadora de lugar utilizando a função SetWindowPos, passando o novo posicionamento. E para finalizar, na procedure HabDesabClick desativamos a janela do bloco de notas. O importante nesse pequeno exemplo é que estamos manipulando uma janela que não pertence ao nosso sistema, ou seja, teoricamente estaria fora de nosso controle. Contudo, através das APIs da categoria Windows podemos acessá-la. Imagine a situação onde um usuário está utilizando seu sistema e ao mesmo tempo está com várias outras janelas abertas: para

DEVMEDIA Copyright - Proibido copiar ou distribuir. Todos os direitos reservados para DevMedia

Listagem 5. Implementando os Códigos dos Botões

```
procedure TFrm_FWindows.Btn_ShowWinClick(Sender: TObject);
var
 Janela: HWND;
begin
 janela := FindWindow(Nil, Pchar('Calculadora'));
 ShowWindow(Janela, SW_SHOWNormal);
end;

procedure TFrm_FWindows.Btn_JanelaClick(Sender: TObject);
var
 Tela: Integer;
 Tamanho: Integer;
 TextoWindows: Array [0..255] of Char;
 Retorno: Integer;
begin
 TextoWindows := '';
 Tela      := GetForegroundWindow();
 Tamanho    := GetWindowTextLength(Tela) + 1;
 Retorno    := GetWindowText(Tela, TextoWindows, Tamanho);
 If Retorno > 0 then
  ShowMessage(TextoWindows);
end;

procedure TFrm_FWindows.Btn_EstJanelaClick(Sender: TObject);
var
 Minimizado: Boolean;
 Maximizado: Boolean;
 WindowNotp: hWnd;
begin
 WindowNotp := FindWindow(Nil, 'Sem título - Bloco de notas'); //Bloco de notas
 Minimizado := IsIconic(WindowNotp);
 Maximizado := IsZoomed(WindowNotp);

 If Minimizado Then
  ShowMessage('Bloco de Notas Minimizado')
 Else
 if  Maximizado  Then
  ShowMessage('Bloco de Notas Maximizado')
 Else
  ShowMessage('Bloco de Notas esta Restaurado');
end;

procedure TFrm_FWindows.Btn_MovJanelaClick(Sender: TObject);
var
 Calculadora: Hwnd;
begin
 Calculadora := FindWindow(Nil, 'Calculadora');

 if SetWindowPos(Calculadora, HWND_TOPMOST, 0, 0, 300, 0, 0) then
  Exit;
end;

procedure TFrm_FWindows.Btn_HabDesabClick(Sender: TObject);
var
 WindowEn: hWnd;
begin
 WindowEn := FindWindow(Nil, 'Sem título - Bloco de notas'); //Bloco de notas

 EnableWindow(WindowEn, False); // True Deixa a Janela Ativa(Normal)
end;
```

chamar sua atenção é possível minimizar essas outras e deixar apenas seu sistema em foco.

Cursores

Os Cursores fazem parte de uma categoria que pode facilmente ser confundida com Mouse, que é outra categoria. Por exemplo, quando o cursor exibir uma ampulheta, não é o mouse que a detém, e sim o cursor. Nessa categoria temos as seguintes funções relacionadas a seguir:

• **ShowCursor:** mostra ou esconde o cursor do mouse. Na verdade, é um contador e, se for 1 então o cursor é visível, se esse contador for negativo, então o cursor não será visível. A função retorna o valor deste contador e possui o parâmetro Bshow, que se for zero, diminui o contador em 1, caso contrário, incrementa em 1.

• **GetCursor**: encontra o identificador para o cursor do mouse em uso atualmente. Esse é o cursor que está sendo usado para representar o ponteiro do mouse na tela. A função retorna um identificador para a imagem se bem-sucedido, ou retornará zero se algum erro ocorrer.

• **GetCursorPos**: lê a posição atual do cursor do mouse, ou seja, as coordenadas X e Y do cursor em relação à tela. Essas informações são transferidas para o parâmetro LpPoint. A função retorna zero se ocorreu um erro ou 1 se for bem-sucedida.

• **LoadCursor:** Carrega um cursor a partir de um arquivo de recurso do programa ou de um arquivo de recurso de cursor do próprio Windows, que pode ser referenciado pelo seu nome ou pelo seu número de ID. Se tudo estiver certo, a função retorna um identificador para o cursor carregado, caso contrário, a função retorna zero. Seus parâmetros são:

- HInstance – Carrega o cursor a partir de um arquivo de recurso do programa. Pode definir zero caso queira carregar de um arquivo de recurso do Windows.
- LpCursorName - Informe o nome ou o número do cursor, para que seja carregado através, de um arquivo de recurso do Windows. Para cursores do Windows, use uma das opções para carregar o cursor desejado:
  - IDC_APPSTARTING - O cursor inicial (seta e ampulheta).
  - IDC_ARROW - O cursor ponteiro de seta regular.
  - IDC_CROSS - O cursor transversal.
  - IDC_IBEAM - O cursor em forma de I (cursor de edição de texto).
  - IDC_NO - O cursor com círculo com uma barra.
  - IDC_SIZEALL - O cursor de quatro pontas.
  - IDC_SIZENESW - O cursor de duas pontas, apontando para o canto superior direito, e inferior esquerdo.
  - IDC_SIZENS - O cursor de duas pontas, apontando para cima e para baixo.
  - IDC_SIZENWSE - O cursor de duas pontas, apontando para o canto inferior direito, e superior esquerdo.
  - IDC_SIZEWE - O cursor de duas pontas, apontando para a esquerda e para a direita.

Copyright - Proibido copiar ou distribuir. Todos os direitos reservados para DevMedia

- IDC_UPARROW - O cursor de seta para cima.
- IDC_WAIT - O cursor de espera (ampulheta).

• **SetCursor**: define a imagem usada para representar o cursor do mouse. O novo cursor pode ser qualquer um válido que tiver sido criado ou carregado. Se for bem-sucedida, a função retorna um identificador para a imagem do cursor, caso contrário, a função retorna zero. Seu único parâmetro é o HCursor, que representa um identificador válido de cursor.

• **SetCursorPos**: define a posição do cursor do mouse. Se você tentar definir as coordenadas fora da área da tela, por exemplo, se você definir a posição para 700,40 em uma tela de 640x480, o cursor irá até a borda da tela ou retângulo. Os parâmetros dessa função são justamente essas coordenadas X e Y.

Aplicação de Cursores

Em uma nova aplicação do tipo Win32 adicionamos cinco controles TLabel e cinco TButton, como mostra a **Figura 4**. Com a tela já montada, modificamos as propriedades Name do Form1 para Frm_Cursor e os cinco TButon para Btn_ShowCursor, Btn_TrocaCur, Btn_PosMouse, Btn_BuscCursor, e Btn_PosCursor. Ao salvar a aplicação definimos a unit com o nome de Unt_Principal e o projeto como Cursores. A **Listagem 6** mostra a utilização da API.

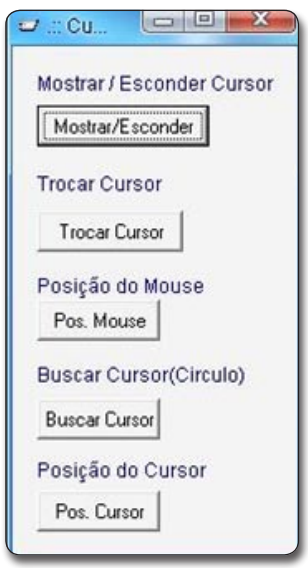


Figura 4. Tela do Aplicativo

]O procedimento Btn_ShowCursorClick simplesmente esconde e exibe o cursor do mouse. Entre uma operação e outra foi utilizado o procedimento sleep, que pausa o processamento do aplicativo em execução por uma quantidade de segundos, expressos em milissegundos. Já o procedimento Btn_TrocaCurClick realiza a troca do cursor. Observe que o cursor em uso é armazenado em uma variável para que possa posteriormente ser recuperado.

Com o procedimento Btn_PosMouseClick recuperamos a posição atual do cursor e a exibimos em tela. No procedimento Btn_BuscCursorClick recuperamos um cursor específico e o definimos como cursor em uso, depois esperamos cinco segundos e desfazemos a operação. Por fim, em Btn_PosCursorClick modificamos a posição do cursor para 30, 30 pixels.

Listagem 6. Implementando os Códigos dos Botões

```
procedure TFrm_Cursor.Btn_ShowCursorClick(Sender: TObject);
begin
 ShowCursor(False);
 Sleep(7000);
 ShowCursor(True);
end;

procedure TFrm_Cursor.Btn_TrocaCurClick(Sender: TObject);
var
 CursorAnt: Integer;
 CursorNov: Integer;
begin
 CursorAnt := GetCursor();
 CursorNov := LoadCursor(0, IDC_SIZEALL);
 SetCursor(CursorNov);
 Sleep (5000);
 SetCursor(CursorAnt);
end;

procedure TFrm_Cursor.Btn_PosMouseClick(Sender: TObject);
var
 Cord: TPoint;
begin
 GetCursorPos(Cord);
 Showmessage('O Mouse esta na Posição X ' + IntTostr(Cord.X) +' Posição Y ' +
IntTostr(Cord.Y));
end;

procedure TFrm_Cursor.Btn_BuscCursorClick(Sender: TObject);
var
 BusCursor: Integer;
 NovCursor: Integer;
begin
 BusCursor := LoadCursor(0, IDC_NO);
 NovCursor := SetCursor(BusCursor);
 Sleep (5000);
 SetCursor(NovCursor);
end;

procedure TFrm_Cursor.Btn_PosCursorClick(Sender: TObject);
begin
 SetCursorPos(30, 30);
end;
```

Registro

Podemos dizer que várias configurações do Sistema Operacional Windows se encontram gravadas no "Registro do Windows". O registro do Windows mantém essas configurações em uma espécie de dicionário, onde temo o par chave e valor. A configuração em si é a chave, e seu conteúdo é o valor. Qualquer mudança realizada em registros existentes deve ser feita com todo o cuidado, porque chave com valor incorreto pode desestabilizar o sistema operacional.

Um exemplo de configuração do Windows armazenada no registro são os dados retornados pela função GetVersionEx, nos traz a versão do Windows instalado no computador. O mesmo poderia ser feito lendo a seguinte chave do registro:

HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\WindowsNT\CurrentVersion\ProductName

DEVMEDIA Copyright - Proibido copiar ou distribuir. Todos os direitos reservados para DevMedia

Então podemos imaginar que, caso modifiquemos essa chave por código, a função GetVersionEx poderia retornar um valor incorreto. A seguir temos funções da categoria Registro.

RegOpenKeyEx

Abre uma chave no registro do Windows. Esta função não irá criar a chave, se ela não existir. A função retorna zero se consegui abrir a chave, ou um código de erro diferente de zero caso não consiga abrir, seus parâmetros são:

- HKey - A chave do registro aberto, ou um dos seguintes valores, sobre o qual a chave está sob:
  - HKEY_CURRENT_USER - Armazena informações sobre os programas do usuário atual.
  - HKEY_LOCAL_MACHINE - Armazena informações sobre os programas para todos os usuários.
  - HKEY_USERS - Contém informações de qualquer usuário, e não apenas o fornecido pela HKEY_CURRENT_USER.
  - HKEY_CURRENT_CONFIG - Armazena informações de configuração do computador.
  - HKEY_DYN_DATA - Armazena dados dinâmicos.
- LpSubKey - O nome da chave para abrir.
- UlOptions - Reservado. Defina como zero.
- SamDesired - Um ou mais dos seguintes valores, especificando o acesso de leitura / gravação desejado:
  - KEY_ALL_ACCESS - Permissão para todos os tipos de acesso.
  - KEY_CREATE_LINK - A permissão para criar links simbólicos.
  - KEY_CREATE_SUB_KEY - Permissão para criar subchaves.
  - KEY_ENUMERATE_SUB_KEYS - Permissão para enumerar subchaves.
  - KEY_EXECUTE - O mesmo que KEY_READ.
  - KEY_NOTIFY - Autorização para prestar notificação de alteração.
  - KEY_QUERY_VALUE - Permissão para consultar dados subchave.
  - KEY_READ - Permissão para o acesso de leitura em geral.
  - KEY_SET_VALUE - Permissão para definir dados subchave.
  - KEY_WRITE - Permissão para o acesso geral de gravação.
- PhkResult - Recebe a informação da chave do registro.

RegCloseKey

RegCloseKey fecha uma chave de registro. Isto deve ser feito depois que terminar de ler ou escrever no registro. Fechando a chave de registro são liberados alguns recursos. Obviamente, você não pode mais usar a chave depois de fechá-la, será preciso abri-la novamente. A função retorna zero se for bem-sucedida, ou um código de erro diferente de zero. Seu único parâmetro é HKey, a chave do registro para fechar.

RegDeleteKey

RegDeleteKey apaga uma chave do registro. A chave a ser apagada não pode ter quaisquer subchaves dentro dela ou então a operação de exclusão falhará. A função retorna zero se for bem-sucedida, ou um código de erro diferente de zero. Seus parâmetros são:

- HKey - Um identificador para uma chave do registro, que é a chave a ser excluída. Ou um dos seguintes valores, especificando a chave do registro:
  - HKEY_CLASSES_ROOT - A chave base de HKEY_CLASSES_ROOT.
  - HKEY_CURRENT_CONFIG - A chave base de HKEY_CURRENT_CONFIG.
  - HKEY_CURRENT_USER - A chave base de HKEY_CURRENT_USER.
  - HKEY_DYN_DATA - A chave base de HKEY_DYN_DATA.
  - HKEY_LOCAL_MACHINE - A chave base de HKEY_LOCAL_MACHINE
  - HKEY_PERFORMANCE_DATA - A chave base de HKEY_PERFORMANCE_DATA.
  - HKEY_USERS - A chave base de HKEY_USERS.
- LpSubKey - O nome da subchave dentro da chave HKey a excluir.

RegDeleteValue

Exclui um valor armazenado em uma chave especificada no registro. Esta função só funciona com valores armazenados, não podendo excluir subchaves. O valor pode ser de qualquer tipo de dado do registo. A função retorna zero se for bem-sucedida, ou um código de erro diferente de zero. Parâmetros:

- HKey - Um identificador para a chave do registro aberto, que contém o valor a ser excluído.
- LpValueName - O nome do valor a ser excluído.

WriteString

WriteString é uma procedure do Delphi e está declarada na unit Registry. O procedimento irá escrever um valor, de qualquer tipo de dado, na chave especificada.

ReadString

Já ReadString é uma função, que também está declarada na unit Registry. Ela vai ler o valor de uma chave, passada como parâmetro, e irá devolver uma string com o valor da chave.

GetValueNames

GetValueNames é uma procedure do Delphi e também está declarada na unit Registry. O procedimento retorna uma lista de strings (Tstrings) de uma chave específica passada como parâmetro.

Aplicação de Registro

Criamos uma nova aplicação Win32 e seu formulário principal adicionamos seis controles TLabel, quatro TButton, dois TMemo e um TImage, como vemos na **Figura 5.** Nosso aplicativa vai configurar a calcular para ser iniciada junto ou não com a inicialização do Windows e irá mostrar os papéis de parede registrados.

Com a tela já montada modificarmos algumas propriedades. A propriedade Name do Form1 para Frm_Registro e os seis

Copyright - Proibido copiar ou distribuir. Todos os direitos reservados para DevMedia

TLabel para Lbl_Iniciar, Lbl_NaoIniciar, Lbl_ImgPapel, Lbl_ItensChave, Lbl_ValorChaves e Lbl_NomeChaves. Os quatro TButton para Btn_Iniciar, Btn_NaoIniciar, Btn_ImgPapel, e Btn_Itens. Vamos agora nomear os nossos dois Memos para Mmo_ValorChaves e Mmo_NomeChaves. Por último vamos nomear o TImage, para Img_PapParede. Ao salvar o projeto, salve a unit com o nome de Unt_Principal e o projeto como (Registro). Agora vamos declarar, duas procedures e uma functions que serão as responsáveis para pegar o caminho da calculadora e por iniciar a mesma junto com o Window ou não. A **Listagem 7** mostra a seção private do formulário.

Depois de fazer estas declarações, basta pressionar simultaneamente as teclas Shift + Ctrl + C e seu código inicial será construído. A **Listagem 8** apresenta a implementação. Não podemos esquecer de adicionar a unit Registry ao uses, pois usaremos seus métodos.

A função DiretorioCalc retorna o diretório de sistema, é nele que se encontra o aplicativo Calculadora. Na sequência temos o procedimento NaoSetAutorum que simplesmente apaga do registro do Windows os dados da calculadora que foram armazenados na seção do registro que inicializa aplicativos. Já o procedimento SetAutorum é responsável por incluir os dados da calculadora nessa seção.

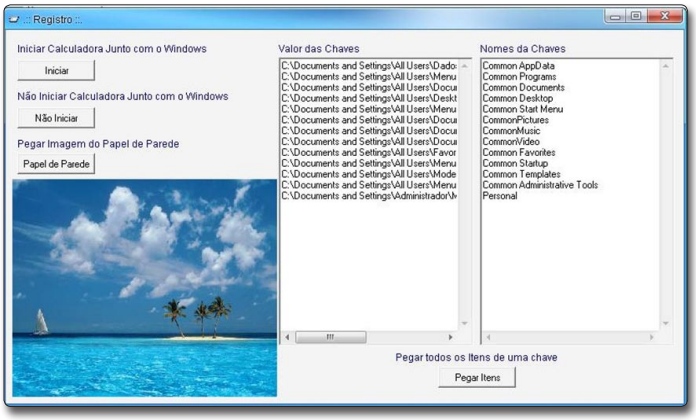


Figura 5. Tela do Aplicativo

Listagem 7. Declaração das funções

```
private
  procedure SetAutorum;
  procedure NaoSetAutorum;
  function DiretorioCalc: String;
  { Private declarations }
```

Listagem 8. Implementando os Códigos do Aplicativo

```
function TFrm_Registro.DiretorioCalc: String;
var
  Buffer: Array[0..255] of Char;
begin
  GetSystemDirectory(Buffer, 255);
  Result := StrPas(Buffer) + '\calc.exe';
end;

procedure TFrm_Registro.NaoSetAutorum;
var
  Reg: TRegistry;
begin
  Reg := TRegistry.Create;

  Try
    Reg.RootKey := HKEY_CURRENT_USER;

    if Reg.OpenKey('\Software\Microsoft\Windows\CurrentVersion\Run', True) then
    begin
      Reg.DeleteValue('Calculadora');
      Reg.DeleteKey('Calculadora');
      Reg.CloseKey;

      Application.MessageBox('Calculadora Não Será Iniciado Com o Windows !',
            ' Atenção', MB_ICONINFORMATION + MB_OK);
    end;

  Finally
    Reg.Free;
    Inherited;
  end;
end;

procedure TFrm_Registro.SetAutorum;
var
  Reg: TRegistry;
begin
  Reg := TRegistry.Create;

  Try
    Reg.RootKey := HKEY_CURRENT_USER;

    if Reg.OpenKey('\Software\Microsoft\Windows\CurrentVersion\Run', True) then
    begin
      Reg.WriteString('Calculadora', DiretorioCalc);
      Reg.CloseKey;

      Application.MessageBox('Calculadora Será Iniciado Junto Com o Windows !',
            ' Atenção', MB_ICONINFORMATION + MB_OK);
    end;

  Finally
    Reg.Free;
    Inherited;
  end;
end;

procedure TFrm_Registro.Btn_IniciarClick(Sender: TObject);
begin
  SetAutorum;
end;

procedure TFrm_Registro.Btn_NaoIniciarClick(Sender: TObject);
begin
  NaoSetAutorum;
end;

procedure TFrm_Registro.Btn_ImgPapelClick(Sender: TObject);
var
  Reg: TRegistry;
begin
  Reg := TRegistry.Create;

  Try
    Reg.RootKey := HKEY_CURRENT_USER;
```


DEVMEDIA Copyright - Proibido copiar ou distribuir. Todos os direitos reservados para DevMedia

Continuação: Listagem 8. Implementando os Códigos do Aplicativo

```
  if Reg.OpenKey('\Control Panel\Desktop', True) then
  begin
   Img_PapParede.Picture.LoadFromFile(Reg.ReadString('Wallpaper'));
   Reg.CloseKey;
  end;

 Finally
  Reg.Free;
  Inherited;
 end;
end;

procedure TFrm_Registro.Btn_ItensClick(Sender: TObject);
var
 Reg: TRegistry;
 Lista: TStrings;
 I: integer;
begin
 Reg := TRegistry.Create;

 Try
  Reg.RootKey := HKEY_LOCAL_MACHINE;

  if Reg.OpenKey('\SOFTWARE\Microsoft\Windows\CurrentVersion\Explorer\
   Shell Folders', True) then
  begin
   Lista := TStringList.Create;
   Reg.GetValueNames(Lista);

   Mmo_NomeChaves.Lines.Add(Lista.Text);

   For I := 0 to Lista.Count -1 do
    Mmo_ValorChaves.Lines.Add(Reg.ReadString(Lista.Strings[I]));

   Lista.Free;
  end;

 Finally
  Reg.CloseKey;
  Reg.Free;
 end;
end;
```

Em ImgPapelClick a imagem selecionada é exibida, a lista de imagens foi obtida por Btn_ItensClick(Sender: TObject).

Como podemos ver, API do Windows pode e deve na auxiliar em diversas tarefas de desenvolvendo de softwares e nos dar uma enorme ajuda em diversos aspectos. Vimos aqui um pouco do seu poder e um pouco de como utilizar todo esse poder.

Existe várias situações em que fazer o uso de API é de uma enorme vantagem. Às vezes, não precisamos fazer enormes rotinas para uma determinada situação, sendo que na API, existe uma função, que faz a mesma situação (objetivo). Então porque não chamar esta função?

Obviamente devemos ter um certo cuidado ao chamar funções que se tratam, diretamente do sistema, arquivos, ou do Registro, enfim categorias destes tipos. Mas aqui, além de aprendermos mais sobre essas categorias, tanto na teoria, como na prática.

Autor

Vanderson Cavalcante de Freitas
vanderson.freitas@ig.com.br
Analista Desenvolvedor Delphi há mais de 5 anos, com experiência em médias e grandes empresas de São Paulo. Formado em técnico em informática no ano de 2003, com diversos cursos em formação específica, como Oracle, Delphi e C#.

Você gostou deste artigo?

Dê seu voto em ***www.devmedia.com.br/clubedelphi/feedback***

Ajude-nos a manter a qualidade da revista!

Copyright - Proibido copiar ou distribuir. Todos os direitos reservados para DevMedia

Cadastros e relatórios dinâmicos em Delphi

Permita ao usuário criar cadastros e relatórios no sistema

Partindo da ideia que um sistema inteligente é aquele que pode ser customizado de acordo com a necessidade do cliente, hoje em dia não temos muitos sistemas inteligentes. Há diversos sistemas grandes, que são construídos com base em um ramo de negócio. Porém, quando é necessário fazer uma simples mudança em um relatório ou em um cadastro, os clientes recebem a reposta que o sistema está estável e que diversas outras empresas usam o mesmo sem problemas. Mas a resposta ideal à sua solicitação seria: vamos adequar, vamos fazer tais mudanças e alterações. Sendo assim é hora de pensar, será que realmente meu negócio está competitivo? Como deve ser a estrutura, da modelagem de dados? Meus cadastros, relatórios e gráficos necessitam constantemente de mudanças?

Então por que não fazer um sistema mais inteligente, ou adicionarmos um módulo ao nosso sistema que permita ao próprio cliente, customizar e criar seus próprios relatórios e ainda indo um pouco mais longe, por que não dar a possibilidade de ele mesmo criar cadastros. Veremos uma forma de como um cliente pode criar cadastros e relatórios personalizados.

Para isso usaremos como banco de dados o Firebird, a ferramenta de relatório será o FastReport, e a parte de conexão de dados usaremos unidac. Lembrando que qualquer que seja o banco de dados, ou qualquer que seja a empresa, seu negócio tem que ser inteligente para que tenha uma maior competitividade.

Fique por Dentro

Empresas que estão ligadas ao desenvolvimento de software preocupam-se cada vez mais em construir softwares que sejam robustos e atendam a todas as necessidades de seus clientes. Sistemas de gestão empresarial, ERP (Enterprise Resource Planning), sempre têm a necessidade de novos módulos, ou adequações para atender um maior público. Módulos como financeiro, administrativo e tantos outros, geralmente possuem seus cadastros e movimentos, e normalmente possuem uma grande quantidade de campos. Por esses e tantos outros motivos, há vários relatórios, com campos diversificados, afim de tornar esses cadastros e movimentos mais eficientes. Este artigo apresenta uma forma de permitir aos próprios usuários a criação de cadastros e relatórios, tornando o sistema muito dinâmico e eficiente em diversos aspectos.

FastReport

O FastReport era uma suíte unicamente externa para geração de relatórios em Delphi. Essa suíte passou a ser incorporada como ferramenta oficial de desenvolvimento de relatórios a partir do Delphi XE2, possuindo uma versão própria para essa finalidade. O FastReport possui algo muito interessante que é a conversões de relatórios Quick Report, Rave Reports e Report Builder por meio de units. É considerado por muitos uma ótima ferramenta de geração de relatórios. Com ele podemos criar desde relatórios simples até os mais complexos. A suíte disponibiliza também o FastScript que permite a criação de scripts em várias linguagens de programação, o FastReport Exports que permite a exportação de relatórios do FastReport para diversos formatos como XLS, HTML, CSV entre outros. Dentre seus vários recursos, da sua versão comercial, usaremos o cross-tab, para criarmos esses relatórios personalizados.

Unidac

O UniDAC provê suporte e acesso a diversos servidores de banco de dados como Oracle, Firebird, InterBase, Microsoft SQL Server, PostgreSQL, MySQL, entre outros. Atende a diversas ferramentas (Delphi, C++ Builder, Lazarus e Free Pascal) em diferentes plataformas (Windows, Mac OS, iOS, Linux e FreeBSD).

Pode se dizer que a estrutura do Unidac é composta por dois elementos. O primeiro deles seria uma engine, ou seja, seu motor que provê ao desenvolvedor uma interface de programação comum e unificada, receptível aos diversos bancos suportados. Já o segundo elemento é a sua parte fundamental, que é a sua camada de acesso a dados. Esse acesso a dados é composto pelos

 Copyright - Proibido copiar ou distribuir. Todos os direitos reservados para DevMedia

provedores (providers), que irão fazer a interação entre a engine e o servidor de banco de dados. Cada provider fica então responsável por trabalhar com um servidor de banco de dados específico. Por exemplo, o TOracleUniProvider para Oracle, TInterBaseUniProvider para InterBase, TPostgreSQLUniProvider para PostgreSQL.

Funcionamento da aplicação

A aplicação permitirá ao usuário criar novos cadastros simples e relatórios simples no sistema sem solicitar uma alteração do sistema ao seu programador. Através de um cadastro o usuário poderá criar a tela, quais campos serão utilizados, etc. Será uma aplicação até relativamente simples. Teremos uma tela principal e teremos um menu, de título Procedimentos, com os seguintes itens: Criar Cadastros, Excluir Cadastros, Sair. Na **Tabela 1** vemos as funcionalidades dos menus.

Menu	Funcionalidade
Criar Cadastro	Irá chamar a tela onde o usuário criará os seus cadastros. Informando o nome da tabela, os seus campos e quais campos irão aparecer em relatório. Terá imagens e textos informando ao usuário como fazer todo o procedimento de criação.
Excluir Cadastro	Irá chamar a tela onde terá uma lista dos cadastros criados. Para que o usuário possa selecionar um cadastro a ser excluído.
Sair	Fecha a nossa aplicação.

Tabela 1. Funcionalidades dos menus

Ao lado deste menu Procedimento teremos um outro menu, Cadastros, e obviamente os itens deste menu, serão os cadastros criados. Por exemplo, caso sejam criados dois cadastros: Clientes e Ordem de Serviço, então esses seriam os itens deste menu.

Ao clicar sobre um menu desses, abrirá a tela do referente cadastro. Nesta tela terá uma grade de dados utilizando um DbGrid, onde será feita a inclusão, alteração e exclusão de dados. Para facilitar toda manipulação de dados, será utilizado um DbNavigator. Também haverá o botão imprimir, que chamará o relatório que foi criado junto com o cadastro, escolhendo quais campos serão impressos.

Enfim, será um cadastro com relatório, com várias consistências, verificações de erros e tudo mais, para que possa ter uma boa usabilidade.

Criando o banco de dados

Para começar, vamos fazer a criação do banco de dados da nossa aplicação. Como é possível ver na **Figura 1**, teremos uma modelagem de dados simples baseada em apenas duas tabelas. A tabela TABELA_USUPER é onde são cadastradas todas as tabelas que o usuário criou. Já a tabela TAB_CAMPOS, é onde são cadastradas todas as informações das tabelas. Informações como nome do campo, apelido do campo, se irá aparecer na grade de dados e no relatório, qual o tipo do campo (Texto, Número, Moeda, Data, Hora, Data e Hora, Observação), e se o campo deverá ou não aparecer no relatório. Para essa tabela criamos também um Generator e uma Trigger, que serão os responsáveis para incrementar o campo código a cada novo registro.

Para o nosso banco de dados foi adotada a versão 2.1 do Firebird. O script SQL para a criação do banco encontra-se exibido na **Listagem 1**.

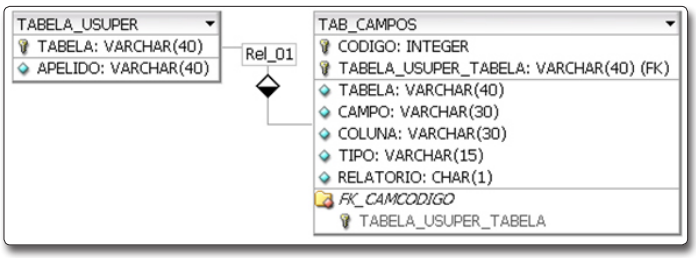


Figura 1. Tabelas do Banco de Dados

Listagem 1. Script de criação do banco de dados

```
SET SQL DIALECT 3;

SET NAMES WIN1252;

CREATE DATABASE '<DIRETÓRIO DO BANCO DE DADOS>\ARTBI.fdb'
  USER 'SYSDBA' PASSWORD 'masterkey'
  PAGE_SIZE 16384
  DEFAULT CHARACTER SET WIN1252;

CREATE GENERATOR GEN_TAB_CAMPOS;

CREATE TABLE TAB_CAMPOS (
  CODIGO       INTEGER NOT NULL,
  TABELA       VARCHAR(40) NOT NULL,
  CAMPO        VARCHAR(30) NOT NULL,
  COLUNA       VARCHAR(30),
  TIPO          VARCHAR(15),
  RELATORIO   CHAR(1)
);
CREATE TABLE TABELA_USUPER (
  TABELA   VARCHAR(40) NOT NULL,
  APELIDO VARCHAR(40)
);

ALTER TABLE TABELA_USUPER ADD CONSTRAINT PK_TABUSERCODIGO
  PRIMARY KEY (TABELA);

ALTER TABLE TAB_CAMPOS ADD CONSTRAINT PK_TABCODIGO
  PRIMARY KEY (CODIGO);

ALTER TABLE TAB_CAMPOS ADD CONSTRAINT FK_CAMCODIGO
  FOREIGN KEY (TABELA) REFERENCES TABELA_USUPER (TABELA)
  ON DELETE CASCADE ON UPDATE CASCADE;

SET TERM ^ ;

CREATE TRIGGER NEW_TAB_CAMPOS FOR TAB_CAMPOS
  ACTIVE BEFORE INSERT POSITION 0
  AS
  begin
    IF (NEW.CODIGO IS NULL) THEN
      NEW.CODIGO = GEN_ID(GEN_TAB_CAMPOS, 1);
  end
  ^

SET TERM ; ^
```

Copyright - Proibido copiar ou distribuir. Todos os direitos reservados para DevMedia

Criamos uma estrutura no banco de dados para armazenar de forma adequada os dados de novos cadastros. Temos a tabela TAB_CAMPOS que armazena os campos de uma determinada tela e suas características e a tabela TABELA_USUPER que armazena as tabelas que precisarão ser criadas no banco de dados para armazenar os dados do novo cadastro.

Criando a aplicação

No Delphi criamos um novo projeto Win32 e salvamos a unit principal com o nome de Unt_Principal.pas e o formulário como Frm_Principal. O projeto salvamos como CriarCadastro ou conforme o gosto. Adicionamos a seguir um novo formulário, salvando-o como Unt_CriaCadastro e o nomeando-o como Frm_CriaCadastro. Agora repetindo o processo, adicionamos mais um formulário e salvamos como Unt_ExcCadastro e o nomeamos como Frm_ExcCadastro. Finalizando a criação dos formulários, adicionamos o último formulário e salvamos sua unit como Unt_Cadastro, e Frm_Cadastro. Na **Tabela 2** identificamos qual será a funcionalidade dos formulários.

Formulário	Funcionalidade
Frm_Principal	Formulário principal da aplicação, onde ficam os menus que chamam todos os outros formulários (Telas).
Frm_CriaCadastro	Pode ser considerado o formulário mais importante, é onde será criado os cadastros e relatórios.
Frm_ExcCadastro	Será o formulário que apresentará todos os cadastrados criados, para que possam ser excluídos.
Frm_Cadastro	E por fim o nosso formulário do cadastro, onde o usuário irá cadastrar, manipular os seus dados e chamar o seu relatório.

Tabela 2. Funcionalidades dos formulários

Programando o formulário principal

Adicionamos então aos formulários os seguintes componentes para conexão: TuniConnection, Provider: TinterBaseUniProvider, Transação: TuniTransaction, Script: TuniScript, Qry_Tabelas: TuniQuery, Qry_Codigo: TuniQuery. Adicionamos também outros três componentes: Mnu_Principal: TmainMenu, Imgl_Menu: TimageList, ApeErro: TapplicationEvents. Com isso o nosso formulário principal fica pronto para ser programado. Após a adição de todos os componentes, ele deverá ficar com a aparência da **Figura 2**.

Nota

Mais adiante vamos utilizar um TClientDataSet, então é importante adicionarmos MidasLib à seção uses, após a interface. Com isso não é necessário distribuir o arquivo Midas.dll.

Na seção private do nosso formulário teremos também declarado, três variáveis e uma procedure. Já na seção public teremos duas procedures e, a seguir, na seção uses do implementation, como vamos usar todos os outros formulários, então fazemos referência a eles, como mostra a **Listagem 2**.

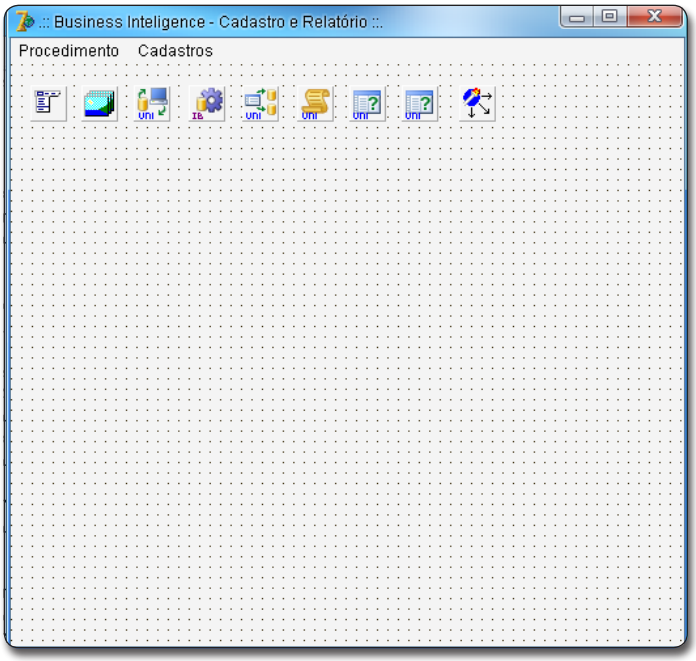


Figura 2. Formulário principal

Listagem 2. Private, Public e Uses do formulário Principal

```
private
  MenuCad,
  MenuTabPer: TMenuItem;
  ImgItMenu: Integer;

  procedure MenuCadPerClick(Sender: TObject);
  { private declarations }
public
  procedure AdicionaMenu(Menu: String);
  procedure RemoveMenu(Menu: String);
  { public declarations }
end;

var
  Frm_Principal: TFrm_Principal;

implementation

Uses Unt_CriaCadastro, Unt_ExcCadastro, Unt_Cadastro;
```

Programaremos então os itens do menu principal, presente na **Figura 3**. O primeiro item do menu, Criar Cadastro, é onde iremos criar o formulário de criação de cadastros. Vamos chamarmos o cadastro e após isso o liberamos, como mostra a **Listagem 3**.

Figura 3. Menu da Aplicação

DEVMEDIA Copyright - Proibido copiar ou distribuir. Todos os direitos reservados para DevMedia

Para o segundo item do menu chamaremos o formulário com os cadastros criados, para que possamos realizar alguma exclusão, como mostra a **Listagem 4**.

Por fim, no menu Sair fecharemos nossa aplicação, como mostra a **Listagem 5**.

Listagem 3. Item do menu Criar Cadastro

```
procedure TFrm_Principal.MnuI_CriaCadClick(Sender: TObject);
begin
 try
  if not Assigned ( Frm_CriaCadastro ) then
   Frm_CriaCadastro := TFrm_CriaCadastro.Create(Self);

  Frm_CriaCadastro.ShowModal;
 finally
  FreeAndNil(Frm_CriaCadastro);
 end;
end;
```

Listagem 4. Item do menu Excluir Cadastro

```
procedure TFrm_Principal.MnuI_ExcCadClick(Sender: TObject);
begin
 try
  if not Assigned ( Frm_ExcCadastro ) then
   Frm_ExcCadastro := TFrm_ExcCadastro.Create(Self);

  Frm_ExcCadastro.ShowModal;
 finally
  FreeAndNil(Frm_ExcCadastro);
 end;
end;
```

Listagem 5. Item do menu Sair

```
procedure TFrm_Principal.MnuI_SairClick(Sender: TObject);
begin
 Application.Terminate;
end;
```

Na **Listagem 6** implementamos a procedure AdicionaMenu, que é a responsável por adicionar no menu (Cadastros) os cadastros criados pelo usuário. Essa procedure recebe como parâmetro o nome do menu a ser criado e é utilizada no momento em que são recuperados os cadastros criados. Essa recuperação ocorre no evento Create do formulário principal.

A **Listagem 7** apresenta como são trazidos do banco de dados cada cadastro criado. Temos duas variáveis declaradas (Path, Banco), que serão as responsáveis por indicar o caminho do nosso banco de dados. Após fazermos a conexão com o banco de dados, buscamos todos os cadastros criados que se encontram na tabela TABELA_USUPER. Então, para cada registro encontrado chamamos a procedure AdicionaMenu, para a criação do menu desse cadastro.

Para o procedimento de exclusão, ao selecionar um cadastro no formulário de exclusão de cadastro (Frm_ExcCadastro), excluímos o mesmo e avisamos ao formulário principal que é necessário remover o item de menu associado a esse registro. Isso é feito pela procedure RemoveMenu, presente na **Listagem 8,** que recebe como parâmetro o nome do menu a ser removido.

Listagem 6. Implementação da procedure AdicionaMenu

```
procedure TFrm_Principal.AdicionaMenu(Menu: String);
var
 I: Integer;
begin
 if ImgItMenu = 0 then
  ImgItMenu := 1
 else
  ImgItMenu := 0;

 for I := 0 to Mnu_Principal.Items.Count - 1 do
 begin
  if AnsiSameCaption(Mnu_Principal.items[I].Caption, 'C&adastros') then
  begin
   MenuCad := Mnu_Principal.items[I];
   Break;
  end;
 end;

MenuTabPer := TMenuItem.Create(MenuCad);
MenuTabPer.Caption := Menu;
MenuTabPer.ImageIndex := ImgItMenu;
MenuTabPer.OnClick := MenuCadPerClick;

 for I := 0 to MenuCad.count - 1 do
 begin
  if MenuCad.Items[I].isLine then
  begin
   MenuCad.Insert(I, MenuTabPer);
   Break;
  end;
 end;
end;
```

Listagem 7. Procedure Create do Form

```
procedure TFrm_Principal.formCreate(Sender: TObject);
var
 Path,
 Banco: String;
begin
 try
  Path  := ExtractFiledir(paramstr(0));
  Banco := Path+'\ARTBI.FDB';
  Conexao.Database := Banco;
  Conexao.Open;
 except
  on E:exception do
  begin
   Application.MessageBox(pansichar('Erro de Conexão'+#13+E.Message),
   ' Atenção', MB_OK + MB_ICONHAND);
   Application.Terminate;
  end;
 end;

 ImgItMenu := 0;

 with Qry_Tabelas do
 begin
  Close;
  SQL.Clear;
  SQL.Add('SELECT * FROM TABELA_USUPER');
  Open;
 end;

 while not(Qry_Tabelas.Eof) do
 begin
  AdicionaMenu(Qry_Tabelas.FieldByName('TABELA').AsString +' - '+
  Qry_Tabelas.FieldByName('APELIDO').AsString);
  Qry_Tabelas.Next;
 end;
end;
```

Copyright - Proibido copiar ou distribuir. Todos os direitos reservados para DevMedia

Na **Listagem 7** quando criamos e adicionamos um menu, configuramos que seu evento Click é implementado pela procedure MenuCadPerClick. Essa procedure cria o formulário do cadastro e passa para ele qual será o cadastro a ser criado, passando para ele qual tabela a ser carregada. Toda manipulação do cadastro é apresentada na **Listagem 9**.

Listagem 8. Procedure RemoveMenu

```
procedure TFrm_Principal.RemoveMenu(Menu: String);
var
 I: Integer;
begin
 for I := 0 to Mnu_Principal.Items.Count - 1 do
 begin
  if AnsiSameCaption(Mnu_Principal.items[I].Caption,'C&adastros') then
  begin
   MenuCad := Mnu_Principal.items[I];
   Break;
  end;
 end;

 for I := 0 to MenuCad.Count - 1 do
 begin
  if AnsiSameCaption(MenuCad.Items[I].Caption, Menu) then
  begin
   MenuCad.Remove(MenuCad.items[I]);
   Break;
  end;
 end;
end;
```

Listagem 9. Click dos Menus dos Cadastrados Criados

```
procedure TFrm_Principal.MenuCadPerClick(Sender: TObject);
begin
 try
  if not Assigned ( Frm_Cadastro ) then
   Frm_Cadastro := TFrm_Cadastro.Create(Self);

  Frm_Cadastro.Tabela := TMenuItem(Sender).Caption;
  Frm_Cadastro.ShowModal;
 finally
  FreeAndNil(Frm_Cadastro);
 end;
end;
```

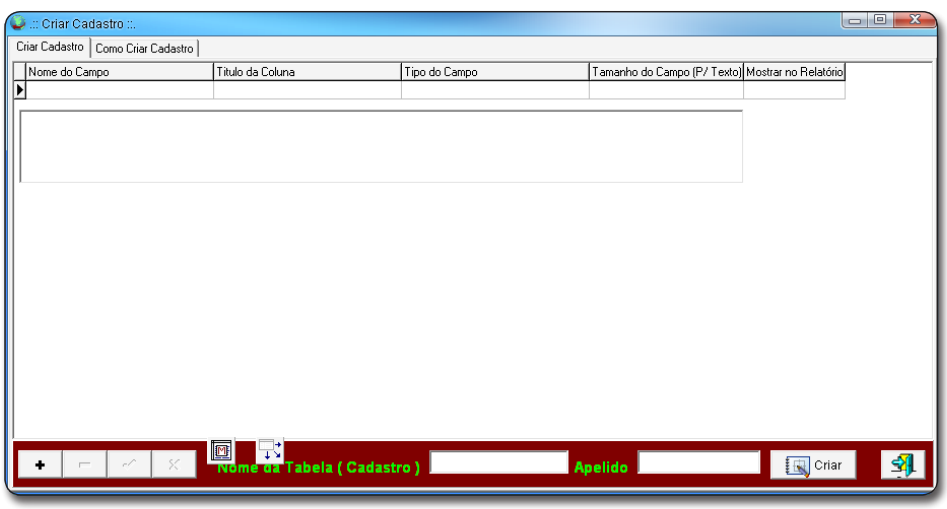


Figura 4. Form de Criação de Cadastros

O tratamento de erros foi todo centralizado através componente TApplicationEvents. Nele colocamos mensagens personalizadas para erros, como "is not a valid date", "Input value", "Insufficient memory for this operation", etc. O componente TApplication-Events pode capturar os eventos da aplicação e um desses eventos é o evento de exceção, ou seja, sempre que uma exceção for levantada ela passará por esse evento, que é onde realizamos toda a tratativa. Fazemos isso conforme a **Listagem 10**.

A partir desse momento vamos tratar da criação das tabelas no banco de dados. Os procedimentos a seguir são essenciais para o projeto e embora possam parecer complexos, com a devida atenção, não o são.

Na seção private do formulário Frm_CriaCadatro temos oito procedures e quatro functions, como na **Tabela 3**, onde é possível identificar o nome e a finalidade de cada uma delas.

Montando o formuláro de criação de cadastros

Agora que já temos conhecimento das funções e procedimentos do formulário de criação de cadastro, iremos montar o formulário, onde temos um TpageControl com duas abas. A primeira aba é utilizada para a criação do cadastro em si. Já a segunda, para ensinar o usuário a fazer os seus cadastros.

Nesta segunda aba temos três imagens. A primeira é um exemplo de criação de um cadastro, a segunda, é a imagem desse cadastro em execução, ou seja, o resultado como ficaria o cadastro criado a partir da primeira imagem, já a terceira imagem seria o relatório desse cadastro. Veremos então esse formulário com foco na primeira aba, a aba de criação, como mostra a **Figura 4**.

Vamos agora aos nossos componentes. Temos um TPanel com alinhamento allbottom. Neste nosso panel temos dois TEdits (Edt_Tabela, Edt_Apelido), junto com dois TLabel que são os títulos dos dois edits. Temos também um TDBNavigator (Nvg_Setas) e temos dois TBitBtn (Btn_Criar, Btn_Sair).

Temos a nossa grade de criação, onde irão ser informados os campos do cadastro. A nossa grade TDBGrid (Grd_Cadastro) está alinhada em toda a área da tela (allclient). E por fim os três últimos componentes, um TMemo (Mmo_Script) que é onde será montado o script para a criação da tabela no banco de dados, esse nosso memo está invisível, ou seja, visible = false. Um TDataSource (Ds_Dados), e um TClientDataSet (Dados), que é onde terão os campos a serem informados para a criação do cadastro. Serão um total de cinco campos, (NOME, COLUNA, TIPOCAMPO, TAMANHO, RELATORIO), todos do tipo TStringField.

Na segunda aba temos três imagens, mostrando um exemplo para a criação de um cadastro, como vemos nas **Figuras 5 a 7.**

DEVMEDIA Copyright - Proibido copiar ou distribuir. Todos os direitos reservados para DevMedia

Nome do Campo	Titulo da Coluna	Tipo do Campo	Tamanho do Campo (P/ Texto)	Mostrar no Relatório
ANIMAL	Animal	Texto	50	S
RACA	Raça	Texto	30	N
DIA	Dia da Consulta	Data	40	S
CONSULTA	Consulta	Hora	40	S
VALORCON	Valor Consulta	Moeda	40	S
OBSANIMAL	Observações	Observação	40	N

Figura 5. Exemplo para a criação de um cadastro de Consulta Veterinária

Procedimento / Função	Finalidade
procedure Replace_Campos;	O script para a criação da tabela, no banco de dados, é montado em um memo (Mmo_Script). Essa procedure troca os textos para o tipo do campo. Ex: (Data para DATE, Texto para VARCHAR, Moeda para DOUBLE PRECISION, Hora para TIME, etc.)
procedure Add_Texto;	Adiciona para o Mmo_Script um campo do tipo texto junto com o seu tamanho informado. Ex: CIDADE VARCHAR (80)
procedure Add_Camps;	Adiciona para o Mmo_Script, os demais tipos de campos, que não precisam de nenhuma outra informação, como o caso do texto, que tem que se informar um tamanho, para o mesmo. Ex: TIME, DATE, TIMESTAMP
procedure MontaScript;	Monta o script para ser executado depois. Cria uma sentença CREATE TABLE + o nome da tabela informada. Logo em seguida cria a chave primária da tabela ('CODIGO CHAR(6) not NULL). Então percorre todos os campos que o usuário informou. Esses campos que estão num ClientDataSet (Dados: TclientDataSet). Os campos são adicionados ao Mmo_Script. Se for campo do tipo texto, é utilizado Add_Texto, se não o Add_Camps. Por último determina que o campo (CODIGO) vai ser a chave primária e usa o Replace_Campos, para terminar a finalização do Script.
procedure CriaTabela;	Aqui então é a criação da tabela, o conteúdo do Mmo_Script é transferido para o componente de script do formulário principal (Script: TuniScript). Então executará o mesmo para que seja criada essa tabela.
procedure InserirTabela;	Neste procedimento é onde criamos o cadastro em si. Aqui inserimos o nome do nosso cadastro, junto com o seu apelido na tabela (TABELA_USUPER). Com o nome do nosso cadastro já incluído junto com o seu apelido (Titulo), cadastramos todos os campos referente a esse cadastro. Para isso, percorremos todos os campos cadastrados no nosso TclientDataSet (Dados) e inserimos esses campos na nossa tabela TAB_CAMPOS. Os campos dessa nossa tabela são: CODIGO: Campo do tipo inteiro, é a chave primária da nossa tabela, informamos o mesmo com a função RetornaCodigo; TABELA: Aqui é o nome da nossa tabela e não o seu apelido (Titulo); CAMPO: O nome do campo (interno), e não o título que será mostrado na grade e no relatório; COLUNA: É o título do campo, aquele que será mostrado na grade e no relatório; TIPO: Qual é o tipo do campo (Texto, Numero, Data, Hora, Data e Hora, Observação, Moeda); RELATORIO: Só aceita dois valores (S / N) para controlar se o campo deverá ou não aparecer no relatório.
procedure ArrumarNomes;	Este procedimento remove os espaços em branco do nome da tabela e dos campos. E também remove os caracteres acentuados.
procedure LimpaGrade;	Aqui apenas fazemos algumas limpezas. Apagamos todos os dados no TclientDataSet (Dados). Limpamos o conteúdo do nosso script de criação (Mmo_Script). E apagamos o nome da tabela e o seu apelido.
function VerConsistencias: Boolean;	Esta função verifica algumas coisas para que não haja erro na hora de criarmos nossa tabela. A verificação começa ao chamarmos o procedimento (ArrumarNomes), para que o script fique adequado. Em seguida é verificado se já não existe um cadastro com o mesmo nome, se foi informado os tipos dos campos, se foi informado o tamanho do campo no caso se o tipo for texto, etc.
function RetornaCodigo(Generator: string): Integer;	Esta função retorna um inteiro que selecionamos do nosso banco de dados, para que nossa chave primária seja um valor único.
function RemoveAcentos(Texto: String):String;	Uma função que irá remover todos os acentos dos caracteres, retornando o texto sem os caracteres acentuados.
function RemoveEspaco(Texto: String):String;	Já esta função irá tirar os espaços em brancos, contidos nos textos.

Tabela 3. Entendendo um pouco a finalidade das funções e procedimentos

Copyright - Proibido copiar ou distribuir. Todos os direitos reservados para DevMedia

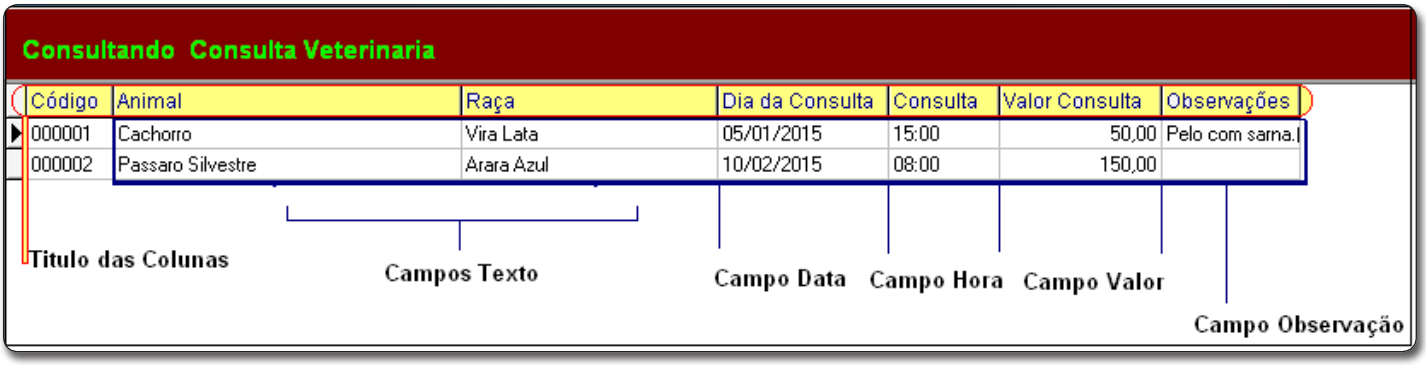


Figura 6. Resultado do cadastro de Consulta Veterinária

Listagem 10. Mensagens personalizadas, para erro ou falhas da aplicação

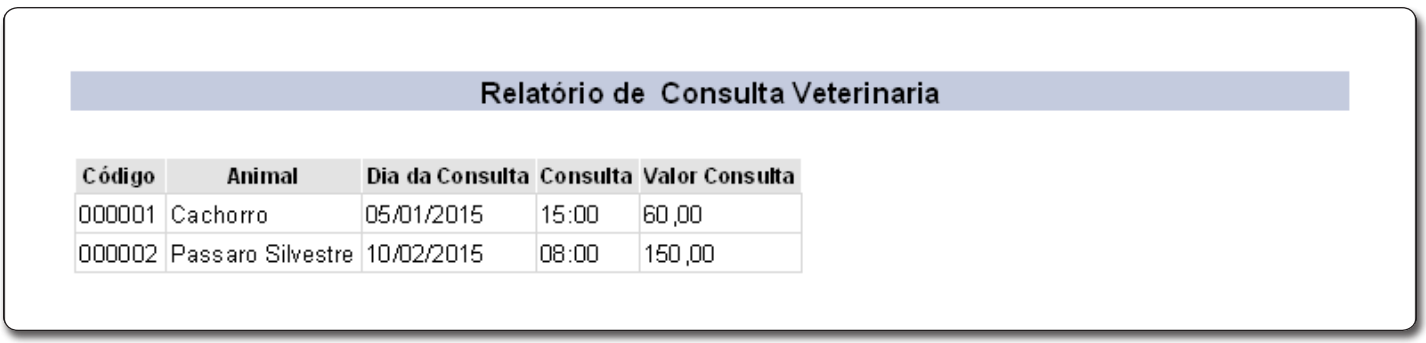
```
procedure TFrm_Principal.ApeErroexception(Sender: TObject; E: exception);
var
  Mensagem: String;
  Pos1, Pos2: Integer;
begin
  if Pos('is not a valid date', E.Message) > 0 then
    Application.MessageBox('Data Invalida !',
   'Atenção', MB_OK + MB_ICONWARNING )
  else
  if Pos('is not a valid time', E.Message) > 0 then
    Application.MessageBox('Hora Invalida !',
   'Atenção', MB_OK + MB_ICONWARNING )
  else
  if Pos('Cannot perform this operation on an empty dataset', E.Message) > 0 then
   Application.MessageBox('Sem Dados para excluir!',
   'Atenção', MB_OK + MB_ICONWARNING )
  else
  if Pos('Invalid input value', E.Message) > 0 then
   Application.MessageBox('Informe o Campo Corretamente!',
   'Atenção', MB_OK + MB_ICONWARNING )
  else
  if Pos('Input value', E.Message) > 0 then
   Application.MessageBox('Informe o Campo Corretamente!',
   'Atenção', MB_OK + MB_ICONWARNING )
  else
  if Pos('Erro ApplyUpdates', E.Message) > 0 then
   Application.MessageBox('Erro ao Gravar no Banco de Dados!',
   'Atenção', MB_OK + MB_ICONWARNING )
  else
  if Pos(UpperCase('is not a valid float'), UpperCase(E.Message)) > 0 then
  begin
    Pos1    := Pos('"', E.Message);
    Mensagem := E.Message;
    Delete(Mensagem, Pos1, 1);
    Pos2    := Pos('"', mensagem);
    mensagem := Copy(E.Message, Pos1 + 1, Pos2 - Pos1);
    mensagem := 'O valor '+ mensagem + ' não é válido.';

    Application.MessageBox(pansichar(mensagem),
   'Atenção', MB_OK + MB_ICONWARNING )
  end
  else
  if Pos('Dataset not in edit or insert mode', E.Message) > 0 then
   Application.MessageBox('Tabela Não está em modo de edição!',
   'Atenção', MB_OK + MB_ICONWARNING )
  else
  if Pos('Error creating cursor handle', E.Message) > 0 then
   Application.MessageBox('Operação realizada com Sucesso !',
   'Parabéns', MB_OK + MB_ICONWARNING )
  else
  if pos(' No current record', E.message)> 0 then
   Application.MessageBox('Nenhum Registro Atual !',
   'Atenção', MB_OK + MB_ICONWARNING )
  else
  if Pos('File Delete operation failed', E.Message) > 0 then
   Application.MessageBox('Falha na Operação de Exclusão de Arquivo! !',
   'Atenção', MB_OK + MB_ICONWARNING )
  else
  if Pos('Access to table disabled because of previous error', E.Message) > 0 then
   Application.MessageBox('Acesso à Tabela desativado por causa de Erro Anterior!',
   'Atenção', MB_OK + MB_ICONWARNING )
  else
  if Pos('Insufficient memory for this operation', E.Message) > 0 then
   Application.MessageBox('Memória Insuficiente para esta Operação!',
   'Atenção', MB_OK + MB_ICONWARNING )
  else
  if Pos('Insufficient disk space', E.Message) > 0 then
   Application.MessageBox('Espaço em disco Insuficiente!',
   'Atenção', MB_OK + MB_ICONWARNING )
  else
  if Pos('Invalid table delete request', E.Message) > 0 then
   Application.MessageBox('Pedido de Apagar inválido!',
   'Atenção', MB_OK + MB_ICONWARNING )
  else
  if Pos('not enough memory', E.Message) > 0 then
   Application.MessageBox('Memória Insuficiente!',
   'Atenção', MB_OK + MB_ICONWARNING )
  else
  if Pos('Table is open', E.Message) > 0 then
   Application.MessageBox('Tabela Está Aberta!',
   'Atenção', MB_OK + MB_ICONWARNING )
  else
  if Pos('Socket Error # 10061 Connection refused', E.Message) > 0 then
   Application.MessageBox('Erro de Conexão!',
   'Atenção', MB_OK + MB_ICONWARNING )
  else
  if Pos('Socket Error # 10060', E.Message) > 0 then
   Application.MessageBox('Erro de Conexão!',
   'Atenção', MB_OK + MB_ICONWARNING )
  else
  if Pos('Socket Error # 11001 Host not found', E.Message) > 0 then
   Application.MessageBox('Erro No Host!',
   'Atenção', MB_OK + MB_ICONWARNING )
  else
  if (Copy(E.Message, 1, 27)= 'Access violation at address') then
  begin
    Application.MessageBox('Ocorreu erro de Violação de Acesso',
   'Atenção', MB_OK + MB_ICONWARNING )
  end
  else
  begin
    Mensagem := 'Ocorreu o seguinte erro: ' + #13 +UpperCase(E.Message);

    Application.MessageBox(pansichar(mensagem),
   'Atenção', MB_OK + MB_ICONWARNING )
  end;
end;

end.
```


DEVMEDIA Copyright - Proibido copiar ou distribuir. Todos os direitos reservados para DevMedia

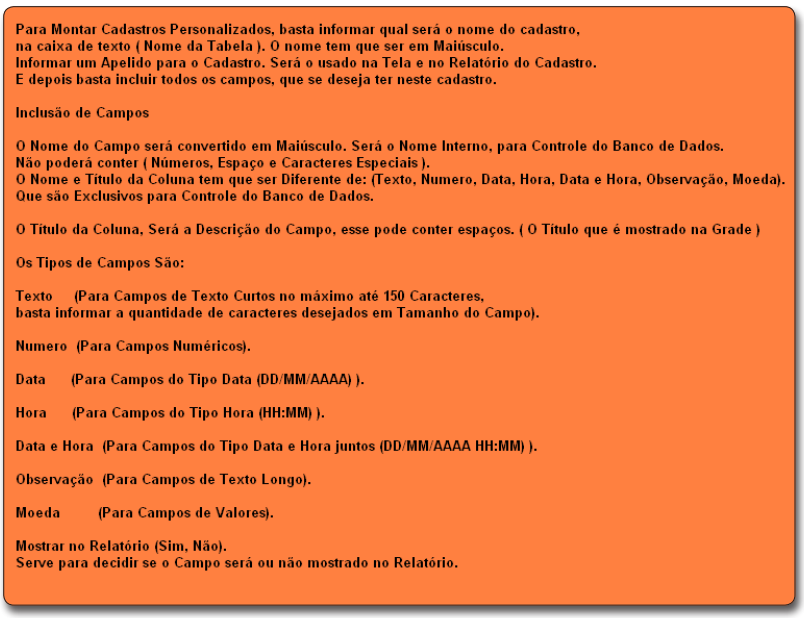


Relatório de Consulta Veterinaria

Código	Animal	Dia da Consulta	Consulta	Valor Consulta
000001	Cachorro	05/01/2015	15:00	60,00
000002	Passaro Silvestre	10/02/2015	08:00	150,00

Figura 7. O Relatório da Consulta Veterinária, conforme campos escolhidos a serem mostrados

Nesta segunda aba ainda temos um TBitBtn, localizado abaixo dessas três imagens e irá mostrar um texto que explica um pouco mais como montar esse cadastro. Esse texto está contido em uma imagem (**Figura 8**) que é exibida ao clicar no TBitBtn.



Para Montar Cadastros Personalizados, basta informar qual será o nome do cadastro,
na caixa de texto (Nome da Tabela). O nome tem que ser em Maiúsculo.
Informar um Apelido para o Cadastro. Será o usado na Tela e no Relatório do Cadastro.
E depois basta incluir todos os campos, que se deseja ter neste cadastro.

Inclusão de Campos

O Nome do Campo será convertido em Maiúsculo. Será o Nome Interno, para Controle do Banco de Dados.
Não poderá conter (Números, Espaço e Caracteres Especiais).
O Nome e Título da Coluna tem que ser Diferente de: (Texto, Numero, Data, Hora, Data e Hora, Observação, Moeda).
Que são Exclusivos para Controle do Banco de Dados.

O Título da Coluna, Será a Descrição do Campo, esse pode conter espaços. (O Título que é mostrado na Grade)

Os Tipos de Campos São:

Texto (Para Campos de Texto Curtos no máximo até 150 Caracteres,
basta informar a quantidade de caracteres desejados em Tamanho do Campo).

Numero (Para Campos Numéricos).

Data (Para Campos do Tipo Data (DD/MM/AAAA)).

Hora (Para Campos do Tipo Hora (HH:MM)).

Data e Hora (Para Campos do Tipo Data e Hora juntos (DD/MM/AAAA HH:MM)).

Observação (Para Campos de Texto Longo).

Moeda (Para Campos de Valores).

Mostrar no Relatório (Sim, Não).
Serve para decidir se o Campo será ou não mostrado no Relatório.

Figura 8. Imagem que contem mais informações de como criar o cadastro

Configurando o TClientDataSet e a grade

Vamos agora inserir os campos para a criação no nosso TClientDataSet e na nossa grade. Para isso, basta dar um duplo clique no TClientDataSet (Dados) e clicar com o botão direito, escolher a opção (New Field), como na **Figura 9**.

Ao fazer isto será exibida uma tela para colocarmos as informações do campo, como vemos na **Figura 10**. Esse processo deve ser repetido cinco vezes, para os nossos campos (NOME, COLUNA, TIPOCAMPO, TAMANHO, RELATORIO). Na tela de informações do campo basta colocar apenas duas informações, o Name que será o nome do campo e o Type que será o tipo do campo, todos os cincos serão do tipo String. Ao finalizar é necessário ativar o nosso TClientDataSet Dados, então clicamos com o botão direito sobre ele e escolhemos a opção Create DataSet,

Como já ativamos o nosso dataset, agora vamos fazer a ligação do nosso TDataSource a esse TDataSet. Então no nosso Ds_Dados indicamos na propriedade DataSet o TClientDataSet Dados. Agora ligamos o nosso Grid (Grd_Cadastro) e o nosso Navegador (Nvg_Setas) a esse TDataSource. Para isso basta selecionarmos os dois componentes e colocar na propriedade DataSource, o nosso Ds_Dados.

Perceba que automaticamente ao ligarmos a grid ao datasource, todos os cincos campos que criamos já aparecem no grid. Agora vamos configurar essa nossa grid. Então selecionamos a mesma e clicamos na propriedade Columns. Será apresentada uma janela que é a janela das colunas, então na parte superior desta clicamos no segundo botão (Add All Fields). Isso faz que os campos sejam listados nesta janela.

Selecionamos então o campo TIPOCAMPO e clicamos na propriedade PickList do mesmo. Feito isto será apresentada a janela para informamos a lista que esse campo deve conter. No nosso caso serão os tipos dos campos, então basta informar os tipos Texto, Numero, Data, Hora, Data e Hora, Observação, Moeda. Lembrando que é uma listagem, então informe um em cada linha, totalizando então sete linhas.

Programando a criação dos cadastros

É preciso garantir uma boa usabilidade e segurança para o usuário no momento da criação de seu cadastro, assim controlamos algumas situações por código. Por exemplo, a **Listagem 11** mostra como impedir a exclusão ou inclusão de registros na grade de campos através de teclas especiais.

Listagem 11. Evento KeyDown da Grade

```
procedure TFrm_CriaCadastro.Grd_CadastroKeydown(Sender: TObject;
  var Key: Word; Shift: TShiftState);
begin
  if (Shift =[ssctrl]) and (key = vk_delete) then abort;
  if (key = vk_Up)then abort;
  if (key = vk_down)then abort;
  if (key = vk_Cancel)then abort;
  if (key = vk_Escape)then abort;
  if (key = vk_Insert)then abort;
end;
```

Copyright - Proibido copiar ou distribuir. Todos os direitos reservados para DevMedia

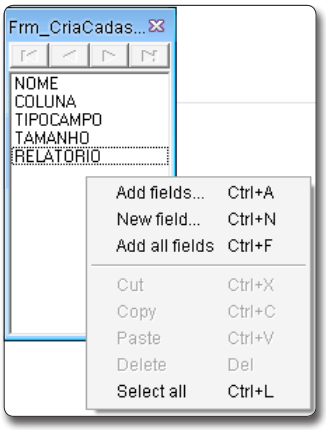


Figura 9. Criando os campos no TClientDataSet (Dados), escolhendo New Field

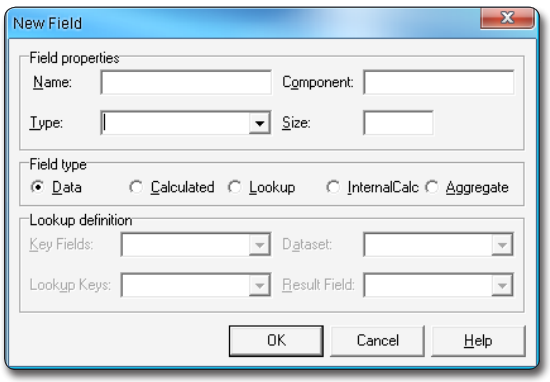


Figura 10. Tela para colocarmos as informações dos campos

Listagem 12. Evento ColExit da Grade

```
procedure TFrm_CriaCadastro.Grd_CadastroColExit(Sender: TObject);
var
 Texto: String;
begin
 if not (Ds_Dados.DataSet.State in [DsEdit, DsInsert]) then
   Exit;

 if (Grd_Cadastro.SelectedIndex = 2) then
 begin
  Texto := Grd_Cadastro.Columns[2].Field.Text;

  if ((Texto <> 'Texto')
  and (Texto <> 'Numero')
  and (Texto <> 'Moeda')
  and (Texto <> 'Data')
  and (Texto <> 'Hora')
  and (Texto <> 'Data e Hora')
  and (Texto <> 'Observação')) then
    Grd_Cadastro.Columns[2].Field.Text := 'Texto';
 end
 else
 if (Grd_Cadastro.SelectedIndex = 4) then
 begin
  Texto := Grd_Cadastro.Columns[4].Field.Text;

  if ((Texto <> 'S') and (Texto <> 'N')) then
    Grd_Cadastro.Columns[4].Field.Text := 'S';
 end;
end;
```

A **Listagem 12** verifica se na coluna 2 foi selecionado um tipo de campo da lista. Se não foi é o tipo do campo como Texto. Já para coluna 4 é verificado se informou ou não se o campo deverá aparecer no relatório. Se não informou, o campo será mostrado no relatório.

Para isso é de extrema importância que os campos sejam criados como mencionado anteriormente, nesta ordem: NOME, COLUNA, TIPOCAMPO, TAMANHO, RELATORIO.

A **Listagem 13** mostra a implementação do evento NewRecord do TClientDataSet, assim, a cada registro novo o campo RELATORIO é configurado com o valor padrão 'S', ou seja, para que ele seja exibido no relatório, da mesma forma o campo TIPOCAMPO tem seu valor padrão como 'Texto' e o tamanho, campo TAMANHO, definido como '40'.

Também é preciso validar o nome da tabela a ser criada, vamos permitir que só contenha letras de A à Z em minúsculo, como mostra a **Listagem 14**, onde verificamos se a tecla pressionada não é "a", "z", ou BackSpace, então soamos o beep e ignoramos a tecla.

Listagem 13. Evento NewRecord do TClientDataSet

```
procedure TFrm_CriaCadastro.DadosNewRecord(DataSet: TDataSet);
begin
 Dados.FieldByName('RELATORIO').AsString := 'S';
 Dados.FieldByName('TIPOCAMPO').AsString := 'Texto';
 Dados.FieldByName('TAMANHO').AsString  := '40';
end;
```

Listagem 14. Evento KeyPress do Edt_Tabela

```
procedure TFrm_CriaCadastro.Edt_TabelaKeyPress(Sender: TObject;
 var Key: Char);
begin
 if not(key in['a'..'z', #8] ) then
 begin
  beep;
  key:=#0;
 end;
end;
```

A **Listagem 15** mostra a implementação do botão de criação (Btn_Criar), apesar de parecer um pouco mais complicada, com um pouco de atenção fica fácil o seu entendimento.

Usamos um bloco try/except para capturar qualquer exceção, para não deixar que uma apareça na tela do usuário. No try mudamos o cursor do mouse, caso o processo demore muito. Verificamos se foi informado o nome da tabela, caso não, retornamos ao cursor normal, e informamos ao usuário. A mesma coisa já acontece logo a seguir, verificamos se foram informados os campos do cadastro, caso não, agimos da mesma forma acima, voltamos ao cursor normal e também informamos o usuário.

Na sequência usamos a função de verificar consistências para ver se podemos prosseguir com o processo de criação. Se estiver tudo certo, aí usaremos quatro funções MontaScript, CriaTabela, InserirTabela, LimpaGrade, para finalizarmos a

DEVMEDIA Copyright - Proibido copiar ou distribuir. Todos os direitos reservados para DevMedia

criação. E então informamos o usuário que o cadastro e o relatório foram criados com sucesso. Caso haja algum erro no meio deste processo todo, entramos no Except. O Except apenas volta o cursor do mouse ao normal e informa ao usuário que houve um erro na criação, informando a mensagem do erro.

Listagem 15. Botão de Criação do Cadastro

```
procedure TFrm_CriaCadastro.Btn_CriarClick(Sender: TObject);
begin
 try
  Screen.Cursor := crSQLWait;

  if (Trim(Edt_Tabela.Text) ='') then
  begin
   Screen.Cursor := crDefault;

   Application.MessageBox('Informe o Nome do Cadastro!!!',
  ' Atenção', MB_OK + MB_ICONINFORMATION);
  Exit;
  end;

  if (Dados.IsEmpty) then
  begin
   Screen.Cursor := crDefault;

   Application.MessageBox('Insira os Campos do Cadastro na Grade!!!',
  ' Atenção', MB_OK + MB_ICONINFORMATION);
  Exit;
  end;

  if not (VerConsistencias) then
  begin
   Screen.Cursor := crDefault;
  Exit;
  end;

  MontaScript;
  CriaTabela;
  InserirTabela;
  LimpaGrade;

  Screen.Cursor := crDefault;

  Application.MessageBox('Cadastro e Relatório Personalizado Criado
  com Sucesso!!!',
 ' Atenção', MB_OK + MB_ICONINFORMATION);

  except
   on E:exception do
   begin
    Screen.Cursor := crDefault;

    Application.MessageBox(PAnsiChar('Erro Ao Criar Tabela:' +#13+ E.message),
    'Business Inteligence', MB_OK + MB_ICONERROR);
  Exit;
  end;
 end;
end;
```

Procedimentos e funções da seção Private

Declaramos na seção Private os procedimentos e funções vistas na **Listagem 16**. Elas realizam operações de apoio, como já foi explicado na **Tabela 3**. Uma vez declaradas, para implementá-las basta pressionar Shift+Ctrl+C.

Nota

No código disponível para download é possível verificar a implementação de todas, aqui no artigo destacamos as mais importantes.

Como o script para a criação da tabela no banco de dados é montado em um memo (Mmo_Script) a procedure Replace_Campos (**Listagem 17**) tem a responsabilidade de trocar os textos que representam tipos em tipos reais. Por exemplo, Data para DATE, Texto para VARCHAR, Moeda para DOUBLE PRECISION, Hora para TIME, Etc.). Repare que ele troca Data por DATE, e Hora por TIME. Em seguida ele verifica se for DATE e TIM) para então trocar por TIMESTAMP.

Listagem 16. Seção Private do formulário

```
 private
  procedure Replace_Campos;
  procedure Add_Texto;
  procedure Add_Camps;
  procedure MontaScript;
  procedure CriaTabela;
  procedure InserirTabela;
  procedure ArrumarNomes;
  procedure LimpaGrade;
  function VerConsistencias: Boolean;
  function RetornaCodigo(Generator: string): Integer;
  function RemoveAcentos(Texto: String):String;
  function RemoveEspaco(Texto: String ):String;
  { private declarations }
```

Listagem 17. Procedure Replace_Campos

```
procedure TFrm_CriaCadastro.Replace_Campos;
begin
 Mmo_Script.Lines.Text :=  StringReplace(Mmo_Script.Lines.Text, 'Texto',
'VARCHAR', [rfReplaceAll]);
 Mmo_Script.Lines.Text :=  StringReplace(Mmo_Script.Lines.Text, 'Observação',
'BLOB SUB_TYPE 1 SEGMENT SIZE 30', [rfReplaceAll]);
 Mmo_Script.Lines.Text :=  StringReplace(Mmo_Script.Lines.Text, 'Data', 'DATE',
[rfReplaceAll]);
 Mmo_Script.Lines.Text :=  StringReplace(Mmo_Script.Lines.Text, 'Hora', 'TIME',
[rfReplaceAll]);
 Mmo_Script.Lines.Text :=  StringReplace(Mmo_Script.Lines.Text, 'DATE e TIME',
'TIMESTAMP', [rfReplaceAll]);
 Mmo_Script.Lines.Text :=  StringReplace(Mmo_Script.Lines.Text, 'Moeda', 'DOUBLE
PRECISION', [rfReplaceAll]);
 Mmo_Script.Lines.Text :=  StringReplace(Mmo_Script.Lines.Text, 'Numero',
'INTEGER', [rfReplaceAll]);
 Mmo_Script.Lines.Text :=  StringReplace(Mmo_Script.Lines.Text, 'Sim', 'S',
 [rfReplaceAll]);
 Mmo_Script.Lines.Text :=  StringReplace(Mmo_Script.Lines.Text, 'Não', 'N',
[rfReplaceAll]);
end;
```

A **Listagem 18** apresenta o procedimento MontaScript. Ele é responsável por montar o script que é executado. Através de variáveis do tipo string todo o texto é montado. Essa montagem é iniciada com a sentença CREATE TABLE + o nome da tabela informada. Logo em seguida a chave primária da tabela, 'CODIGO CHAR(6) not NULL. Então todos os campos que o usuário informou são percorridos. Esses campos estão no ClientDataSet Dados.

Copyright - Proibido copiar ou distribuir. Todos os direitos reservados para DevMedia

Então um a um é adicionado no nosso Mmo_Script, verificando se for campo do tipo texto, utiliza-se o procedimento Add_Texto, se não for do tipo texto, utiliza-se o procedimento Add_Camps. Para finalizar, é removida a última vírgula que foi adicionada após o último campo, fechando a sentença então com os caracteres “’);” e o AlteraTabela. Então para que o script fique totalmente correto e possa ser executado no banco de dados, é chamado o procedimento Replace_Campos.

Bom agora que estamos com o nosso script correto, então é necessário criar a tabela. A **Listagem 19** mostra o método CriaTabela, ele irá jogar o conteúdo do Mmo_Script no componente de script do formulário principal (Script: TuniScript) e então executará o mesmo, obviamente avisando se houve alguma falha.

Listagem 18. Procedure MontaScript

```
procedure TFrm_CriaCadastro.MontaScript;
var
 CriaTabela,
 AlteraTabela,
 NomeTabela,
 UltLinha,
 Codigo: String;
 I, Virgula: Integer;
begin
 try
  NomeTabela := Edt_Tabela.Text;

  CriaTabela :='CREATE TABLE ' + NomeTabela + ' (';

  Codigo := 'CODIGO CHAR(6) not NULL,';

  AlteraTabela :='ALTER TABLE '+ NomeTabela +
  ' ADD CONSTRAINT PK_' + NomeTabela + ' PRIMARY KEY (CODIGO);';

  Dados.First;

  Mmo_Script.Lines.Add(CriaTabela);
  Mmo_Script.Lines.Add(Codigo);

  while not Dados.Eof do
  begin
   if (Dados.FieldByName('TIPOCAMPO').AsString ='Texto')then
    Add_Texto
   else
    Add_Camps;

    Dados.Next;
  end;

  I := Mmo_Script.Lines.Count - 1;
  UltLinha := Mmo_Script.Lines[I];
  Virgula := Pos(',', UltLinha);

  if Virgula > 0 then
  begin
   Delete(UltLinha, Virgula, Length(UltLinha));
   Insert(');', UltLinha, Virgula);
   Mmo_Script.Lines[I]:= UltLinha;
  end;

  Mmo_Script.Lines.Add(AlteraTabela);

  Replace_Campos;
 except
  on E:exception do
  begin
   Screen.Cursor := crDefault;

   Application.MessageBox(PAnsiChar('Erro Ao Montar Script:' +#13+ E.message),
   'Business Inteligence', MB_OK + MB_ICONERROR);
  Exit;
  end;
 end;
end;
```

Listagem 19. Procedure CriaTabela

```
procedure TFrm_CriaCadastro.CriaTabela;
begin
 try
  Frm_Principal.Script.SQL.Text := Mmo_Script.Lines.Text;
  Frm_Principal.Script.Execute;
 except
  on E:exception do
  begin
   Screen.Cursor := crDefault;

   Application.MessageBox(PAnsiChar('Erro Ao Criar Tabela:' +#13+ E.message),
   'Business Inteligence', MB_OK + MB_ICONERROR);
   Exit;
  end;
 end;
end;
```

O procedimento InserirTabela mostrado na **Listagem 20** é responsável por criar o cadastro em si. Aqui inserimos o nome do cadastro junto com o seu apelido na tabela (TABELA_USUPER). Por exemplo, Nome (CONSVET), Apelido (Consulta Veterinária). Posteriormente são cadastrados todos os campos referente a esse cadastro, percorrendo o conteúdo do TClientDataSet Dados. A cada registro encontrado é realizada uma inserção na tabela TAB_CAMPOS. Os campos dessa nossa tabela são:

- CODIGO: campo do tipo inteiro, é a chave primária da tabela, informamos o mesmo com a função RetornaCodigo);
- TABELA: é o nome da tabela e não o seu apelido (Titulo);
- CAMPO: o nome do campo (interno) e não o título que será mostrado na grade e no relatório;
- COLUNA: aqui sim é o título do campo, aquele que será mostrado na grade e no relatório;
- TIPO: indica o tipo do Campo, Texto, Numero, Data, Hora, Data e Hora, Observação, Moeda;
- RELATORIO: Só aceita dois valores (S/N) para controlar se o campo deverá ou não aparecer no relatório.

Feito isto, o nosso Cadastro e Relatório Personalizado foram criados com sucesso. No menu do formulário principal esse cadastro já pode ser acessado como um outro qualquer.

Claro que para permitir a criação de um cadastro e o mesmo possa ser inserido no banco de dados, temos que realizar algumas consistências. A função VerConsistencias da **Listagem 21** mostra isso. O procedimento ArrumarNomes (ver arquivos do download) é executado para que o script fique adequado. Esse procedimento remove caracteres indesejados dos nomes informados.

DEVMEDIA Copyright - Proibido copiar ou distribuir. Todos os direitos reservados para DevMedia

Listagem 20. Procedure InserirTabela

```
procedure TFrm_CriaCadastro.InserirTabela;
var
 Tabela,
 Apelido,
 TpCampo,
 Relatorio: String;
begin
 try
  Tabela  := Trim(Edt_Tabela.Text);
  Apelido := Edt_Apelido.Text;

  if (Apelido = '')then
   Apelido := Tabela;

  with Frm_Principal.Qry_Tabelas do
  begin
   Close;
   SQL.Clear;
   SQL.Add('INSERT INTO TABELA_USUPER (TABELA, APELIDO) VALUES
   (:TABELA, :APELIDO)');
   Params.ParamByName('TABELA').AsString  := Tabela;
   Params.ParamByName('APELIDO').AsString := Apelido;
   Execute;
  end;

  with Frm_Principal.Qry_Tabelas do
  begin
   Close;
   SQL.Clear;
   SQL.Add('INSERT INTO TAB_CAMPOS (CODIGO, TABELA, CAMPO, COLUNA, TIPO,
   RELATORIO) VALUES ');
   SQL.Add('(:CODIGO, :TABELA, :CAMPO, :COLUNA, :TIPO, :RELATORIO)');

   Params.ParamByName('CODIGO').AsInteger := RetornaCodigo
   ('GEN_TAB_CAMPOS');
   Params.ParamByName('TABELA').AsString  := Tabela;
   Params.ParamByName('CAMPO').AsString   := 'CODIGO';
   Params.ParamByName('COLUNA').AsString  := 'Código';
   Params.ParamByName('TIPO').AsString    := 'STRING';
   Params.ParamByName('RELATORIO').AsString := 'S';
   Execute;
  end;

  Dados.First;

  while not Dados.Eof do
  begin
   if (Dados.FieldByName('TIPOCAMPO').AsString ='Texto')then
    TpCampo := 'STRING'
   else
   if (Dados.FieldByName('TIPOCAMPO').AsString ='Numero')then
    TpCampo := 'INTEGER'
   else
   if (Dados.FieldByName('TIPOCAMPO').AsString ='Data')then
    TpCampo := 'DATE'
   else
   if (Dados.FieldByName('TIPOCAMPO').AsString ='Hora')then
    TpCampo := 'TIME'
   else
   if (Dados.FieldByName('TIPOCAMPO').AsString ='Data e Hora')then
    TpCampo := 'TIMESTAMP'
   else
   if (Dados.FieldByName('TIPOCAMPO').AsString ='Observação')then
    TpCampo := 'MEMO'
   else
   if (Dados.FieldByName('TIPOCAMPO').AsString ='Moeda')then
    TpCampo := 'DOUBLE';

   if (Dados.FieldByName('RELATORIO').AsString ='S')then
    Relatorio := 'S'
   else
   if (Dados.FieldByName('RELATORIO').AsString ='N')then
    Relatorio := 'N';

   with Frm_Principal.Qry_Tabelas do
   begin
    Close;
    SQL.Clear;
    SQL.Add('INSERT INTO TAB_CAMPOS (CODIGO, TABELA, CAMPO, COLUNA, TIPO,
    RELATORIO) VALUES ');
    SQL.Add('(:CODIGO, :TABELA, :CAMPO, :COLUNA, :TIPO, :RELATORIO)');

    Params.ParamByName('CODIGO').AsInteger := RetornaCodigo
    ('GEN_TAB_CAMPOS');
    Params.ParamByName('TABELA').AsString  := Tabela;
    Params.ParamByName('CAMPO').AsString   := Dados.FieldByName('NOME').
    AsString;
    Params.ParamByName('COLUNA').AsString  := Dados.FieldByName('COLUNA').
    AsString;
    Params.ParamByName('TIPO').AsString    := TpCampo;
    Params.ParamByName('RELATORIO').AsString := Relatorio;
    Execute;
   end;

   Dados.Next;
  end;

  Frm_Principal.AdicionaMenu(Tabela +' - '+ Apelido);

 except
  on E:exception do
  begin
   Screen.Cursor := crDefault;

   Application.MessageBox(PAnsiChar('Erro Ao Inserir Tabela:' +#13+ E.message),
          'Business Inteligence', MB_OK + MB_ICONERROR);
   Exit;
  end;
 end;
end;
```

Copyright - Proibido copiar ou distribuir. Todos os direitos reservados para DevMedia

Em seguida outras verificações necessárias são realizadas. Por exemplo, se já não existe um cadastro com o mesmo nome, se foi informado o tipo dos campos, se foi informado o tamanho dos campos, no caso se o tipo for texto se foi informado se o campo irá ou não aparecer no relatório.

Montando o formulário dos cadastros

O formulário de cadastro possui sete componentes referente a parte de relatórios. São eles:

• FrxPDF (TfrxPDFExport), usado para exportar o relatório para o formato PDF;

Listagem 21. Function VerConsistencia

```
function TFrm_CriaCadastro.VerConsistencias: Boolean;
begin
 Result := True;

 ArrumarNomes;

 with Frm_Principal.Qry_Tabelas do
 begin
  Close;
  SQL.Clear;
  SQL.Add('SELECT TABELA FROM TABELA_USUPER ');
  SQL.Add('WHERE TABELA =:TABELA');
  Params.ParamByName('TABELA').AsString := Edt_Tabela.Text;
  Open;
 end;

 if not(Frm_Principal.Qry_Tabelas.IsEmpty) then
 begin
  Screen.Cursor := crDefault;

  Application.MessageBox('Já Existe um Cadastro Personalizado com este Nome!!!',
  ' Atenção', MB_OK + MB_ICONINFORMATION);
  Result := False;
  Exit;
 end;

 Dados.First;

 while not Dados.Eof do
 begin
  if (Dados.FieldByName('NOME').AsString ='Texto') or
    (Dados.FieldByName('NOME').AsString ='Numero') or
    (Dados.FieldByName('NOME').AsString ='Data') or
    (Dados.FieldByName('NOME').AsString ='Hora') or
    (Dados.FieldByName('NOME').AsString ='Data e Hora') or
    (Dados.FieldByName('NOME').AsString ='Observação') or
    (Dados.FieldByName('NOME').AsString ='Moeda') or
    (Dados.FieldByName('COLUNA').AsString ='Texto') or
    (Dados.FieldByName('COLUNA').AsString ='Numero') or
    (Dados.FieldByName('COLUNA').AsString ='Data') or
    (Dados.FieldByName('COLUNA').AsString ='Hora') or
    (Dados.FieldByName('COLUNA').AsString ='Data e Hora') or
    (Dados.FieldByName('COLUNA').AsString ='Observação') or
    (Dados.FieldByName('COLUNA').AsString ='Moeda') then
  begin
    Screen.Cursor := crDefault;

    Application.MessageBox(PAnsiChar('Atenção Não Pode Haver:'+#13+
    'Texto, Numero, Data, Hora, Data e Hora, Observação, Moeda.'+#13+
    'No ( Nome do Campo ) e em ( Titulo da Coluna ).'),
    ' Atenção', MB_OK + MB_ICONINFORMATION);
    Result := False;
    Exit;
  end;

  if (Dados.FieldByName('TIPOCAMPO').AsString ='') then
   begin
    Screen.Cursor := crDefault;

    Application.MessageBox('Informe o Tipo de Campo para todos os Campos(Linhas)!!!',
    ' Atenção', MB_OK + MB_ICONINFORMATION);
    Result := False;
   Exit;
   end;

  if (Dados.FieldByName('TIPOCAMPO').AsString ='Texto') and
    (Dados.FieldByName('TAMANHO').AsString ='')then
    begin
    Screen.Cursor := crDefault;

    Application.MessageBox('Informe o Tamanho para o Tipo de Campo Texto!!!',
    ' Atenção', MB_OK + MB_ICONINFORMATION);
    Result := False;
    Exit;
   end;

  if (Dados.FieldByName('RELATORIO').AsString ='')then
    begin
    Screen.Cursor := crDefault;

    Application.MessageBox('Informe se os Campos irão aparecer no Relatório ou não!!!',
    ' Atenção', MB_OK + MB_ICONINFORMATION);
    Result := False;
   Exit;
   end;

  if Pos(' ', Dados.FieldByName('NOME').AsString) > 0 then
    begin
    Screen.Cursor := crDefault;

    Application.MessageBox('Não Pode Ter Espaço em Branco no Campo Nome!!!',
    ' Atenção', MB_OK + MB_ICONINFORMATION);
    Result := False;
   Exit;
   end;

  if Pos(' ', Edt_Tabela.Text) > 0 then
    begin
    Screen.Cursor := crDefault;

    Application.MessageBox('Não Pode Ter Espaço em Branco no Nome do Cadastro!!!',
    ' Atenção', MB_OK + MB_ICONINFORMATION);
    Result := False;
   Exit;
   end;

   Dados.Next;
  end;
 end;
```


DEVMEDIA Copyright - Proibido copiar ou distribuir. Todos os direitos reservados para DevMedia

• FrxHTML (TfrxHTMLExport), usado para exportar o relatório para o formato HTML;
• FrxJPEG (TfrxJPEGExport), usado para exportar o relatório para o formato JPEG;
• FrxEXCEL (TfrxXLSExport), usado para exportar o relatório para o formato XLS;
• FrxCSV (TfrxCSVExport), usado para exportar o relatório para o formato CSV;
• FrxCross (TfrxCrossObject), componente usado para a montagem e exibição dos dados;
• FrxRelPerso (TfrxReport), componente do relatório.

São necessários mais dois componentes não visuais: Qry_Cadastro (TuniQuery) e Ds_Cadastro (TDataSource), que serão os responsáveis para a parte dos dados do cadastro. Temos também dois TPanel. O primeiro painel Pnl_Tabela é alinhado ao topo do formulário, nele conterá um TLabel, Lbl_Status. Esse label servirá, para informar o status do cadastro se está consultando, editando, ou inserindo dados.

O segundo panel (Pnl_Botao) é alinhado na parte de baixo do formulário. Temos nesse painel quatro componentes, dois TDBNavigator (Nvg_Setas e Nvg_Dados). O primeiro exibe os botões de navegação de registros, já o segundo os botões para manipulação de dados. O primeiro navegador só fica habilitado se não estiver inserindo ou editando o cadastro. Os outros dois componentes são TBitBtn (Btn_Imprimir e Btn_Sair). O botão de imprimir será responsável por gerar o relatório. Por fim, temos o último componente, o Grd_Cadastro, um TDBGrid. A grade do cadastro que fica alinhada em todo o restante da tela, entre os dois painéis. Montamos esse formulário para que ele fique parecido com a **Figura 11**.

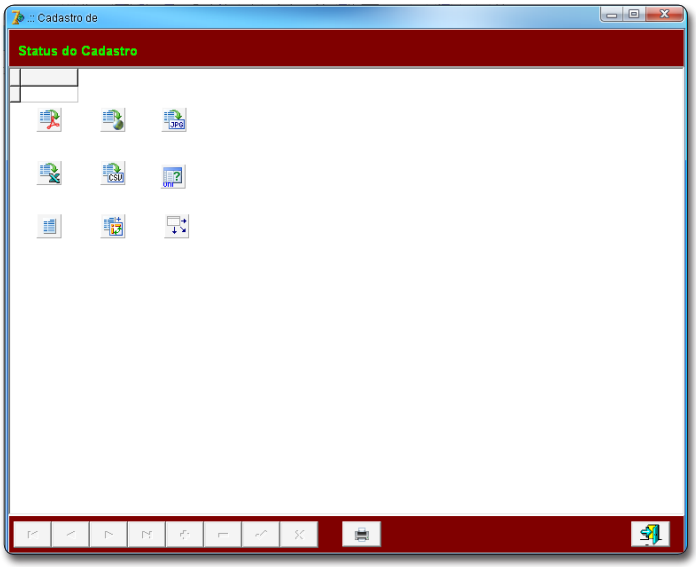


Figura 11. Montagem do Form de Cadastros

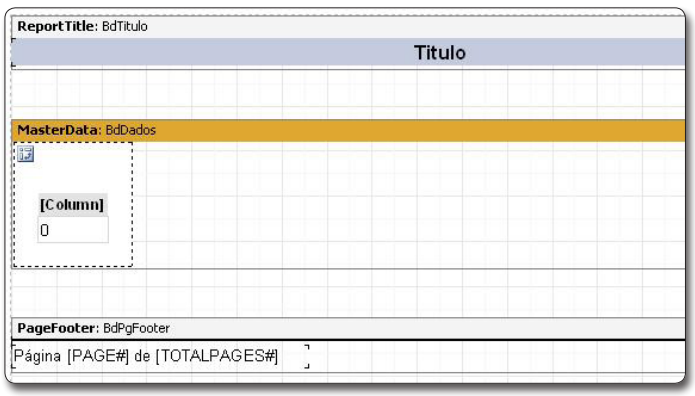


Figura 12. Montagem do Relatório

Criando o Relatório

Para criar o relatório basta dar um clique no componente de relatório FastReport, FrxRelPerso. Mudamos as subpropriedades da propriedade PreviewOptions, nela temos Buttons onde devemos deixar que só esses fiquem habilitados (true) os seguintes botões: pbPrint, pbExport, pbZoom, pbTools, pbNavigator, pbExportQuick, pbNoFullScreen.

Feita esta pequena mudança agora é só dar um duplo clique no componente do relatório que será exibido o seu designer para fazermos sua montagem. No centro fica a página do relatório e a esquerda uma coleção de botões, os quais usaremos alguns para montar o relatório. Nesse nosso relatório vamos ter três bandas, uma para o título, uma para os dados e a outra do rodapé que informará o número da página. A imagem desta montagem pode ser vista na **Figura 12**.

Banda do Título

Aqui é uma simples banda que exibirá o título do relatório, esse título será: Relatório de Apelido do cadastro. Por exemplo: Relatório de Consulta Veterinária. Então clicamos no botão Inserir Banda e escolhemos a banda Título do Relatório, a nomeamos como BdTitulo. Agora nesta banda inserimos um objeto Memo e o nomeamos para MmoTitulo, mudando sua propriedade Align para baWidth e a propriedade HAlign para haCenter.

Banda dos Dados

Clicamos no botão de Inserir banda e escolhemos a banda Dado Mestre. Aparecerá uma tela que informa que a banda não está relacionada com DataSet algum, basta clicar no OK. Nomeie esta banda para BdDados e alteramos a propriedade Height para 2,80. Inserimos nesta banda um objeto CrossTab. Na tela que aparecerá, na parte direita no canto superior, a propriedade Colum, nela escolhemos sem ordenação. Logo a baixo, na propriedade Cell, escolher "Nenhum". Mais um pouco em baixo, nas caixas de seleção, deixar marcado somente as opções "Exibe Canto, Cabeçalho de Coluna, Tamanho Automático, Arredonda Bordas das Células, Reimprime cabeçalho em nova página". Selecione o estilo Gray, ou conforme o gosto. Clique no ok para fechar a tela e vamos fazer as outras mudanças.

Nomeie o objeto CrossTab para Cross, mude a propriedade Top para 0, a propriedade Width para 2,62 e a propriedade Left para

Copyright - Proibido copiar ou distribuir. Todos os direitos reservados para DevMedia

0,05 e a propriedade Height para 2,76. Agora clicamos na primeira coluna, onde está escrito Columm. Mudamos a propriedade HAlign para hacenter e a propriedade VAlign para vacenter. Colocamos sua fonte com o estilo Negrito e mudamos a sua cor para cl3DLight, ou outra desejada.

Agora clicamos na parte de baixo, onde está o 0 (zero). Mudamos a propriedade color para clWhite, a sua propriedade HAlign para haLeft, na propriedade Frame no BottonLine mude a propriedade color para clMenuText.

Banda do Rodapé

Aqui também é uma simples banda, ela informará o número da página e o total de páginas. Clicamos novamente no botão de inserir banda, escolhemos a banda Rodapé de Página, a nomeamos para BdPgFoote). Agora inserimos um objeto Texto, nomeamos para MmoLinhaFooter. Na propriedade Frame mudamos o Width para 2, e seu Top para 0. Também alteramos a propriedade Width para 2, em TopLine, RightLine, LeftLine, BottonLine mudamos a propriedade Type para ftTop como true.

Agora inserimos um objeto (Texto do Sistema), na sua tela que aparecerá marcamos a opção de Texto, que está na parte de baixo da tela. Nela escrevemos o seguinte texto, logo em baixo na sua caixa de texto: (Página [PAGE#] de [TOTALPAGES#]). Agora basta dar ok e nomear o mesmo para SmmoPagina, mudamos também a sua propriedade Top para 0,10 e o Left para 0. Ajustamos o seu tamanho para que caiba todo o texto e aí finalizamos a montagem.

Procedimentos e funções do fomulário dos cadastros

Na seção Uses declaramos o formulário principal e em seguida declaramos uma constante, que será usada para o status do cadastro, conforme mostra a **Listagem 22**.

Listagem 22. Adicionando Units ao Uses, e Declarando uma Constante

```
unit Unt_Cadastro;
interface
Uses
  Unt_Principal, DBAccess, Uni //etc
Const
  dsEditModesStr: array [1..3] of String = ('Consultando', 'Editando', 'Inserindo');
```

Na seção private e public temos algumas variáveis, procedimentos e funções. São elas que irão montar o nosso cadastro e auxiliar em diversas outras rotinas. Olhamos com atenção a seção public, nela está declarada uma variável Tabela. É o formulário principal que irá passar para essa variável qual é o cadastro escolhido e a ser montado. Vejamos como ficam essas seções na **Listagem 23** e sua implementação está disponível no código fonte do artigo.

Programando o formulário de excluir cadastros

O formulário Frm_ExcCadastro é o mais simples todos os outros. São apenas quatro componentes e quatro procedures. Na sua montagem foi utilizado um TPanel (Pnl_Botao) alinhado em baixo da tela AllBottom. Dentro deste Pnl_Botao foram colocados dois TBitBtn (Btn_Excluir, Btn_Sair). Obviamente que o primeiro é para excluir um cadastro e o segundo é para fechar o formulário. E por último um TListBox (Lst_Cadastros) que por sua vez é alinhado em todo o restante da tela (AllClient), nele é onde serão listados os cadastros existentes. Após a sua montagem, ele deverá ficar com a aparência da **Figura 13**.

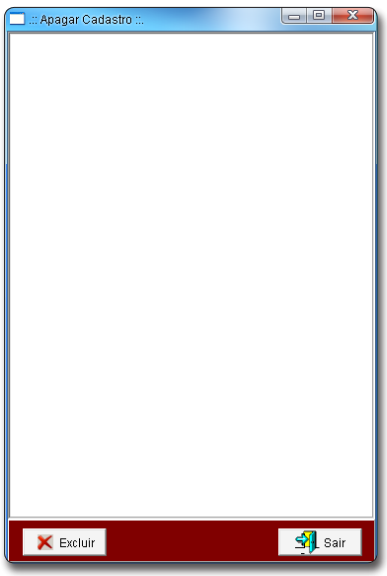


Figura 13. Montagem do Form de Excluir Cadastros

Listagem 23. Seção Private e Public do Formulário

```
private
  Apelido,
  Nome,
  Sql: String;
  Frm: Tform;
  MmoGrade: TMemo;
  ListApelidos: TStringList;
  OldStateCad: TDataSetState;
  procedure MontaCadastro;
  procedure DataHoraText(Sender: TField; const Text: String);
  procedure MemoText(Sender: TField; var Text: String; DisplayText:
    Boolean);
  procedure FecharClick(Sender: TObject);
  procedure ConfirmarClick(Sender: TObject);
  function fZerosLeft(Str: String; Tam: Word): String;
  function fCodDefault(Qry: TUniQuery; Chave, Tab: String; nInc: Integer;
    lZerosLeft: Boolean; Condicao: String = ''; Tabela: TDataSet = nil;
    Edit: TCustomEdit = nil): String;
  { private declarations }
public
  Tabela: String;
  { public declarations }
end;
```

No evento Create do formulário selecionamos todos os campos da tabela TABELA_USUPER, que é onde ficam as informações dos cadastros criados. Percorremos o resultado da consulta e adicionamos no TListBox o Cadastro (TABELA) e o seu apelido (APELIDO), como mostra **Listagem 24**.

DEVMEDIA Copyright - Proibido copiar ou distribuir. Todos os direitos reservados para DevMedia

Listagem 24. Evento Create do Formulário

```
procedure TFrm_ExcCadastro.formCreate(Sender: TObject);
begin
 with Frm_Principal.Qry_Tabelas do
 begin
  Close;
  SQL.Clear;
  SQL.Add('SELECT * FROM TABELA_USUPER');
  Open;
 end;

 while not(Frm_Principal.Qry_Tabelas.Eof) do
 begin
  Lst_Cadastros.Items.Add(Frm_Principal.Qry_Tabelas.FieldByName('TABELA').
  AsString +'-'+
  Frm_Principal.Qry_Tabelas.FieldByName('APELIDO').AsString);

  Frm_Principal.Qry_Tabelas.Next;
 end;

 if not (Frm_Principal.Qry_Tabelas.IsEmpty) then
  Lst_Cadastros.Selected[0] := True;
end;
```

Listagem 25. Botão Excluir Cadastro

```
procedure TFrm_ExcCadastro.Btn_ExcluirClick(Sender: TObject);
begin
 if (Lst_Cadastros.ItemIndex < 0) then
 begin
  Application.MessageBox('Sem Cadastro/Selecionado para Apagar!!!',
  ' Atenção', MB_OK + MB_ICONINFORMATION);
  Exit;
 end;

 if Application.MessageBox('Deseja Realmente Apagar o Cadastro Selecionado??',
  ' Apagar o Cadastro', MB_ICONQUESTION + MB_YESNO) <> idYes then Exit;

 ExcTabela;
end;
```

A **Listagem 25** mostra a exclusão disparada pelo botão Btn_Excluir e que usa a procedure ExcTabela. Essa procedure deve ser declarada na seção private também. Verificamos se existe algum cadastro, caso não, informamos o usuário que não há cadastro para excluir. Se existir o cadastro, é perguntado ao usuário se ele realmente deseja excluir: se sim é chamada a procedure ExcTabela, que pode ser vista na **Listagem 26**.

CURSOS ONLINE

A Revista .net Magazine oferece aos seus assinantes uma série de Cursos Online de alto padrão de qualidade.

.net magazine

CONHEÇA OS CURSOS MAIS RECENTES:

- Curso Padrões de Projeto com C#
- Curso básico de ASP .NET
- Curso de Introdução ao .NET Framework
- Curso Básico de C#
- C# 5 e suas novidades
- ASP.NET MVC – Sistema de Vestibular

Para mais informações :
www.devmedia.com.br/curso/netmagazine
(21) 3382-5038

DEVMEDIA

Listagem 26. Procedimento ExcTabela

```
procedure TFrm_ExcCadastro.ExcTabela;
var
 Tabela,
 TApelido: String;
begin
 try
  TApelido := Lst_Cadastros.Items.Strings[Lst_Cadastros.ItemIndex];
  Tabela  := Copy(TApelido, 0, Pos('-', TApelido )-1);

  Screen.Cursor := crSQLWait;

  with Frm_Principal.Qry_Tabelas do
  begin
   Close;
   SQL.Clear;
   SQL.Add('DELETE FROM TAB_CAMPOS TC ');
   SQL.Add('WHERE (TC.TABELA = :PTABELA)');

   Params.ParamByName('PTABELA').AsString := Tabela;
   Execute;
  end;

  with Frm_Principal.Qry_Tabelas do
  begin
   Close;
   SQL.Clear;
   SQL.Add('DELETE FROM TABELA_USUPER TU ');
   SQL.Add('WHERE (TU.TABELA = :PTABELA)');
   Params.ParamByName('PTABELA').AsString := Tabela;
   Execute;
  end;

  with Frm_Principal.Qry_Tabelas do
  begin
   Close;
   SQL.Clear;
   SQL.Add('DROP TABLE '+ Tabela);
   Execute;
  end;

  Frm_Principal.RemoveMenu(TApelido);
  Lst_Cadastros.Items.Delete(Lst_Cadastros.ItemIndex);
  Screen.Cursor := crDefault;

 except
  on E:exception do
  begin
   Screen.Cursor := crDefault;

   Application.MessageBox(PAnsiChar('Erro Ao Excluir Tabela:' +#13+ E.message),
   'Business Inteligence', MB_OK + MB_ICONERROR);
   Exit;
  end;
 end;
end;
```

Nela, repare que temos duas variáveis (Tabela, TApelido), que servem para pegar o nome e o apelido da tabela a ser excluída. Em seguida apagamos os campos pertencentes a essa tabela, que estão na tabela TAB_CAMPOS. Logo após apagamos o cadastro desta tabela, que se encontra na tabela TABELA_USUPER. Feito isso apagamos essa tabela com o comando Drop Table Agora sim no nosso banco de dados não há mais nada referente a esse cadastro.

Na sequência chamamos a procedure RemoveMenu, passando como parâmetro a variável apelido, e tiramos o cadastro desta lista e do menu.

Permitir que um usuário possa criar cadastros simples e seus respectivos relatórios é uma funcionalidade que garante flexibilidade e pode até mesmo ser um diferencial comercial para qualquer produto.

Autor

Vanderson Cavalcante de Freitas
vanderson.freitas@ig.com.br
Analista Desenvolvedor Delphi há mais de 5 anos, com experiência em médias e grandes empresas de São Paulo. Formado em técnico em informática no ano de 2003, com diversos cursos em formação específica, como Oracle, Delphi e C#.

Você gostou deste artigo?

Dê seu voto em ***www.devmedia.com.br/clubedelphi/feedback***

Ajude-nos a manter a qualidade da revista!

DEVMEDIA Copyright - Proibido copiar ou distribuir. Todos os direitos reservados para DevMedia

RENOVE JÁ!

Sua assinatura pode estar acabando

Renovando a assinatura de sua revista favorita você ganha brindes e descontos exclusivos.

Faça a renovação de sua assinatura agora mesmo.

www.devmedia.com.br/renovacao ou (21)3382-5038

DEVMEDIA

Somos tão apaixonados por tecnologia que o nome da empresa

diz tudo.

Porta 80 é o melhor que a Internet pode oferecer para sua empresa.

Já completamos 8 anos e estamos a caminho dos 80, junto com nossos clientes.

Adoramos tecnologia.
Somos uma equipe composta de gente que entende e gosta do que faz,
assim como você.

Estrutura

100% NACIONAL. Servidores de primeira linha, links de alta capacidade.

Suporte diferenciado

Treinamos nossa equipe para fazer mais e melhor. Muito além do esperado.

Serviços

Oferecemos a tecnologia mais moderna, serviços diferenciados e antenados com as suas necessidades.

1-to-1

Conhecemos nossos clientes. Atendemos cada necessidade de forma única.
Conheça!

Porta 80
WEB HOSTING

Hospedagem | Cloud Computing | Dedicados | VoIP | Ecommerce | Aplicações | Streaming | Email corporativo

porta80.com.br | comercial@porta80.com.br | twitter.com/porta80

SP 4063-8616 | RJ 4063-5092 | MG 4063-8120 | DF 4063-7486